

JuMP Syntax Cheatsheet

Optimization with Julia

This cheatsheet summarizes the most common syntax elements of [JuMP](#) for optimization modeling in Julia. It includes examples for setting up models, declaring variables and constraints, defining objectives, adjusting solver options, solving, and accessing solutions.

Note: This cheatsheet uses the latest JuMP syntax. For more detailed documentation, visit the [JuMP documentation](#).

Model Setup

Importing Packages and Creating a Model

```
using JuMP, HiGHS

# Create a model with HiGHS optimizer
model = Model(HiGHS.Optimizer)

# Set a time limit for the optimizer (optional)
set_optimizer_attribute(model, "time_limit", 60.0)
```

Variables

Declaration

Continuous Variables

```
# Unbounded continuous variable
@variable(model, x)

# Non-negative continuous variable
@variable(model, x >= 0)

# Bounded continuous variable (0 <= x <= 10)
@variable(model, 0 <= x <= 10)

# Fixed variable, x is fixed at 5
@variable(model, x == 5)
```

Integer Variables

```
# Unbounded integer variable
@variable(model, x, Int)

# Non-negative integer variable
@variable(model, x >= 0, Int)

# Bounded integer variable (0 <= x <= 10)
@variable(model, 0 <= x <= 10, Int)
```

Binary Variables

```
# Binary variable (0 or 1)
@variable(model, x, Bin)
```

Containers

Arrays and Matrices

```
# Array of 5 continuous variables
```

```
@variable(model, x[1:5])

# Non-negative array of variables
@variable(model, x[1:5] >= 0)

# Binary array of variables
@variable(model, x[1:5], Bin)

# 3x4 matrix of variables
@variable(model, x[1:3, 1:4])

# Integer 3x4 matrix
@variable(model, x[1:3, 1:4], Int)
```

Custom Indexing

```
indices = ["A", "B", "C"]
@variable(model, x[i in indices])
```

Constraints

Basic Constraints

```
# Declare additional variables as needed
@variable(model, x)
@variable(model, y)

@constraint(model, con1, 2x + y <= 10)
@constraint(model, con2, x + 2y >= 5)
@constraint(model, con3, x == y) # Equality constraint
```

Constraints with Containers

```
# Array Variables
@variable(model, x[1:5] >= 0)

# Constraint for each variable element
@constraint(model, capacity[i=1:5],
    x[i] <= 100
)

# Sum constraint across array
@constraint(model, total_sum,
    sum(x[i] for i in 1:5) <= 500
)

# Matrix constraints: each element gets its constraint
@variable(model, y[1:3, 1:4])
@constraint(model, matrix_con[i=1:3, j=1:4],
    y[i,j] <= i + j
)
```

Conditional Constraints

```
# Constraint applies only for indices where i > 2
@constraint(model, cond[i=1:5; i > 2],
    x[i] <= 10
)
```

```
# Multiple conditions for two-dimensional index:
@constraint(model, cond2[i=1:10, j=1:10; i != j && i + j <= 15],
    x[i,j] + x[j,i] <= 1
)
```

Constraints with Conditional Summations

```
# Sum over a subset of indices (e.g., i > 2)
@constraint(model, total_sum,
    sum(x[i] for i in 1:5 if i > 2) <= 500
)

# Sum with multiple conditions on a 2D array
@constraint(model, total_sum2,
    sum(x[i,j] for i in 1:5, j in 1:5 if i != j && i + j <= 7) <= 1
)
```

Tip: Use semicolons (;) to separate the index definition from conditions when defining constraints.

Objective Functions

Basic Objectives

```
@objective(model, Max, 5x + 3y) # Maximize
@objective(model, Min, 2x + 4y) # Minimize
```

Objectives Using Containers

```
# Container-based objective
@variable(model, z[1:10])
@objective(model, Min, sum(z[i] for i in 1:10))

# Weighted objective function
weights = [1, 2, 3, 4, 5]
@objective(model, Max,
    sum(weights[i] * z[i] for i in 1:5)
)
```


Solver Options

```
# Recreate model with a solver if needed
model = Model(HiGHS.Optimizer)

# Set a 60-second time limit
set_time_limit_sec(model, 60)
println("Current time limit: ", time_limit_sec(model))

# Tolerance attributes: relative and absolute gap
set_optimizer_attribute(model, "mip_rel_gap", 0.01) # 1% gap tolerance
set_optimizer_attribute(model, "mip_abs_gap", 0.1)  # Absolute gap tolerance

# Presolve options: enable or disable
set_optimizer_attribute(model, "presolve", "on") # Enable presolve
# set_optimizer_attribute(model, "presolve", "off") # To disable presolve
```

Tip: Adjust these settings to balance between solution precision and computational speed.

Solving and Inspecting the Model

```
# Optimize the model
optimize!(model)

# Check the termination status using MathOptInterface statuses
status = termination_status(model)
if status == MOI.OPTIMAL
    println("Solution is optimal")
elseif status == MOI.TIME_LIMIT && has_values(model)
    println("Time limit reached with a feasible solution")
else
    println("Solver status: ", status)
end

# Get variable and objective values
x_val = value(x)
println("x value: ", x_val)

# For arrays, retrieve values with broadcasting:
x_vals = value.(x)
println("x array values: ", x_vals)

# Get the objective value
obj_val = objective_value(model)
println("Objective value: ", obj_val)
```

Important: Always check the termination status before using the solution values.

Model Modifications

Variable Updates

```
# Set or update variable bounds
set_lower_bound(x, 0)
set_upper_bound(x, 10)

# Fix x to a specific value, then unfix if needed
fix(x, 5)
unfix(x)
```

Constraint Updates

```
# Delete a constraint if necessary
delete(model, con1)
```

Note: After modifying the model, you will need to re-solve it to update the solution.

Additional Features and Advanced Topics

Checking Variable Properties

```
# Check bounds for a variable
println("Has lower bound? ", has_lower_bound(x))
println("Lower bound: ", lower_bound(x))
println("Has upper bound? ", has_upper_bound(x))
println("Upper bound: ", upper_bound(x))

# Check variable type and retrieve information
println("Is x binary? ", is_binary(x))
println("Is x integer? ", is_integer(x))
println("Variable name: ", name(x))
println("Total number of variables: ", num_variables(model))
```

Simple Model Example

```
# Create a simple model example
using JuMP, HiGHS

model = Model(HiGHS.Optimizer)
@variable(model, 0 <= x <= 10)
@constraint(model, con1, x >= 5)
@objective(model, Max, x)

optimize!(model)

# Introspection on the variable x
println("Has lower bound? ", has_lower_bound(x))
println("Lower bound: ", lower_bound(x))
println("Has upper bound? ", has_upper_bound(x))
println("Upper bound: ", upper_bound(x))
println("Is x binary? ", is_binary(x))
println("Is x integer? ", is_integer(x))
println("Is x fixed? ", is_fixed(x))
println("Variable name: ", name(x))
println("Total number of variables: ", num_variables(model))
```

Running HiGHS 1.8.1 (git hash: 4a7f24ac6): Copyright (c) 2024 HiGHS under MIT licence terms

Coefficient ranges:

Matrix	[1e+00, 1e+00]
Cost	[1e+00, 1e+00]
Bound	[1e+01, 1e+01]
RHS	[5e+00, 5e+00]

Presolving model

0 rows, 0 cols, 0 nonzeros 0s

0 rows, 0 cols, 0 nonzeros 0s

Presolve : Reductions: rows 0(-1); columns 0(-1); elements 0(-1) - Reduced to empty

Solving the original LP from the solution after postsolve

Model status	: Optimal
Objective value	: 1.0000000000e+01
Relative P-D gap	: 0.0000000000e+00
HiGHS run time	: 0.00
Has lower bound?	true
Lower bound:	0.0
Has upper bound?	true
Upper bound:	10.0
Is x binary?	false
Is x integer?	false

```
Is x fixed? false  
Variable name: x  
Total number of variables: 1
```

Key Points

- Always check solution status before using results
- Set appropriate time limits for large problems
- Use gap tolerances to balance precision and speed
- Monitor solve time for performance optimization
- Consider presolve for complex problems