Tutorial III.II - Package Management

Applied Optimization with Julia

Introduction

Welcome to this beginner-friendly guide on understanding packages and package management in Julia!

Think of packages as pre-written sets of tools that extend what Julia can do. It's like having a toolbox where you can add new tools (packages) to help you solve specific problems. For example, there are packages for working with data, creating visualizations, or solving complex math problems.

The best part? Most Julia packages are free to use, thanks to the open-source community!

> **i** Note
>
> While we'll practice some commands here, you'll typically manage packages in Julia's REPL (Read-Eval-Print Loop), which is like Julia's command center.

Section 1 - Using the Package Manager

Julia's built-in package manager, Pkg, provides a robust set of tools for managing packages. To utilize these tools, start by importing the Pkg module. This module allows you to add, update, and remove packages, and manage environments efficiently.

> **i** Note
>
> Note, `import PackageName` lets you access exported functions with `PackageName.function`, while `using PackageName` imports all exported names into the local namespace for direct access.

Exercise 1.1 - Import the Pkg Module

Import the Pkg module to start managing packages effectively.

```
# YOUR CODE BELOW
```

```
# Test your answer
try
    Pkg.update()
```

```
    println("Pkg module imported successfully and packages were updated!")
catch e
    @error "The Pkg module was not imported yet! Have you used the correct
syntax?"
end
```

Section 2 - Managing Environments and Adding Packages

Think of environments as separate workspaces for different projects. It's like having different folders for different school subjects. It's a best practice to create a new environment for each project to avoid conflicts between package versions. Activate a new environment with `Pkg.activate("new_environment")` and add packages to it with `Pkg.add("PackageName")`. This ensures that each of your projects has a clean, dedicated set of dependencies. When you create a new environment, Julia generates two important files:

1. `Project.toml`: Lists direct dependencies of your project.
2. `Manifest.toml`: Contains a complete dependency graph of your project.

These files help ensure reproducibility and should be version controlled with your project. Remember what we did at the start of the lecture? We activated `Pkg.activate("applied-optimization")` together, which means we are already working in an environment!

Before installing new packages, we should always activate this environment again. This ensures that the packages are installed in the environment and not globally.

Remember how we did this at the start?

```
Pkg.activate("applied-optimization")
```

> Tip
>
> Our environment is linked to the Jupyter kernel. It is the thing in the upper right corner we are using in this course. Therefore, we don't need to activate it most of the time as it is already running automatically. But to update packages or add new packages, we should activate it so the packages are added permanently.

> Tip
>
> Once created and while working in the folder with `Project.toml` and `Manifest.toml`, a simple `Pkg.activate()` is faster and enough to load the environment in the working folder!

Adding packages in Julia is afterwards straightforward using the `Pkg.add("PackageName")` function. Replace `PackageName` with the actual package name you wish to add.

Exercise 2.1 - Add the DataFrames Package

Let's add a popular package called DataFrames. It's great for working with structured data, like spreadsheets.

```
# YOUR CODE BELOW
```

```
# Test your answer
try
    using DataFrames
    println("Package added successfully!")
catch e
    @error "Package was not added yet! Have you used the correct syntax?"
end
```

Section 3 - Updating and Removing Packages

Just like apps on your phone, packages can be updated or removed when you no longer need them.

- To update a package: `Pkg.update("PackageName")`
- To update all packages: `Pkg.update()`
- To remove a package: `Pkg.rm("PackageName")`

If you wanted to update DataFrames, you'd use `Pkg.update("DataFrames")`. To remove it, you'd use `Pkg.rm("DataFrames")`.

Conclusion

Congratulations! You've completed the tutorial on packages and package management in Julia. These skills are important for effectively managing and utilizing external libraries. Continue to the next file to learn more.

Solutions

You will likely find solutions to most exercises online. However, I strongly encourage you to work on these exercises independently without searching explicitly for the exact answers to the exercises. Understanding someone else's solution is very different from developing your own. Use the lecture notes and try to solve the exercises on your own. This approach will significantly enhance your learning and problem-solving skills.

Remember, the goal is not just to complete the exercises, but to understand the concepts and improve your programming abilities. If you encounter difficulties, review the lecture materials, experiment with different approaches, and don't hesitate to ask for clarification during class discussions.

Bibliography