Tutorial III.II - Package Management

Applied Optimization with Julia

Introduction

This interactive script serves as an in-depth guide to understanding packages and package management in Julia. Delve into the utilization of the package manager, the processes of adding, updating, and removing packages, and management of distinct environments. First off all, we briefly need to introduce what a package is: a package is a collection of code that is used to extend the functionality of Julia. There are a lot of packages available for Julia, ranging from basic packages like `LinearAlgebra` to more specialized packages like `JuMP` for optimization. This means that most of the time you don't have to invent the wheel by yourself but can rely on the work of others. The great thing: Almost all packages available for Julia are available for free and can be used under the `MIT license`.

> **Note**
>
> Note: While this script offers guidance, package management commands are usually executed in Julia's REPL. The REPL is the command line interface to Julia where you can interact with the language.

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. 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 - Adding Packages

Adding packages in Julia is 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

Use `Pkg.add()` to add the Package `DataFrames` that we will use later.

```
# 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

Maintaining your Julia environment involves keeping your packages up-to-date and removing those that are no longer necessary. Use `Pkg.update("PackageName")` to update individual packages, or simply `Pkg.update()` to update all packages. To remove an obsolete package, use `Pkg.rm("PackageName")`. Remember to replace `PackageName` with the actual name of the package you wish to manage.

Section 4 - Managing Environments

Julia environments are essential for keeping project dependencies isolated and organized. 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. But don't worry, you don't have to worry about this yet.

> **Note**
>
> If you plan to stick to working with notebooks, you don't have to care about environments yet. Just use the regular environment for the course and you don't have to activate anything.

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.

Later, you will find the solutions to these exercises online in the associated GitHub repository, but we will also quickly go over them in next week's tutorial. To access the solutions, click on the Github button on the lower right and search for the folder with today's lecture and tutorial. Alternatively, you can ask ChatGPT or Claude to explain them to you. But please remember, the goal is not just to complete the exercises, but to understand the concepts and improve your programming abilities.