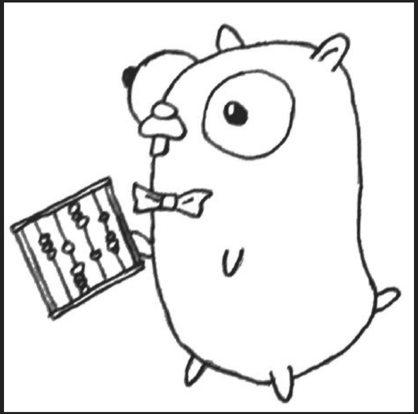
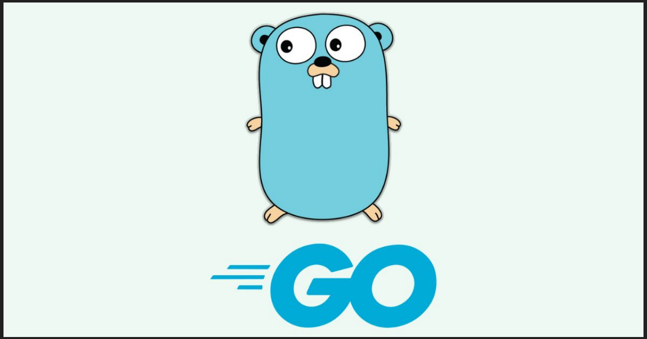
Analyzing the Performance and Accuracy of the GoNum Matrix Library



Cameron Brown



What is GoNum Matrix Library?

The Gonum library is a collection of numerical and scientific computing packages written for the Go programming language. It provides various functionalities, including matrix manipulation, linear algebra, statistical analysis, and optimization, designed to offer high performance and concurrency capabilities.

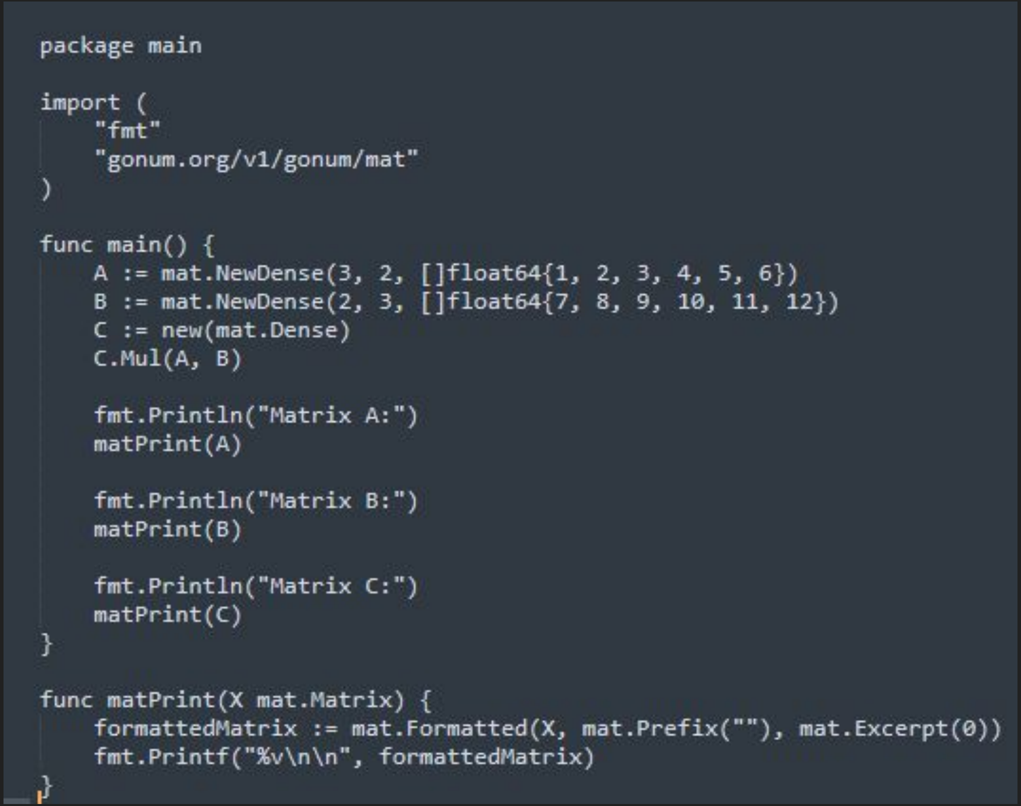
For the interest of our studies, I will be focusing on the Gonum/Mat package that provides Linear Algebra capabilities to the Go language.

Go as a language is well known for being a high performance, simple syntax language and therefore I am interested to see if that is true for it’s linear algebra packages.

Working with GoNum Examples:

GoNum is capable of completing many of the same operations as Julia and NumPy. For example, here is how matrix multiplication is done in Go.

Compare this to Julia and NumPy shown on next slide.

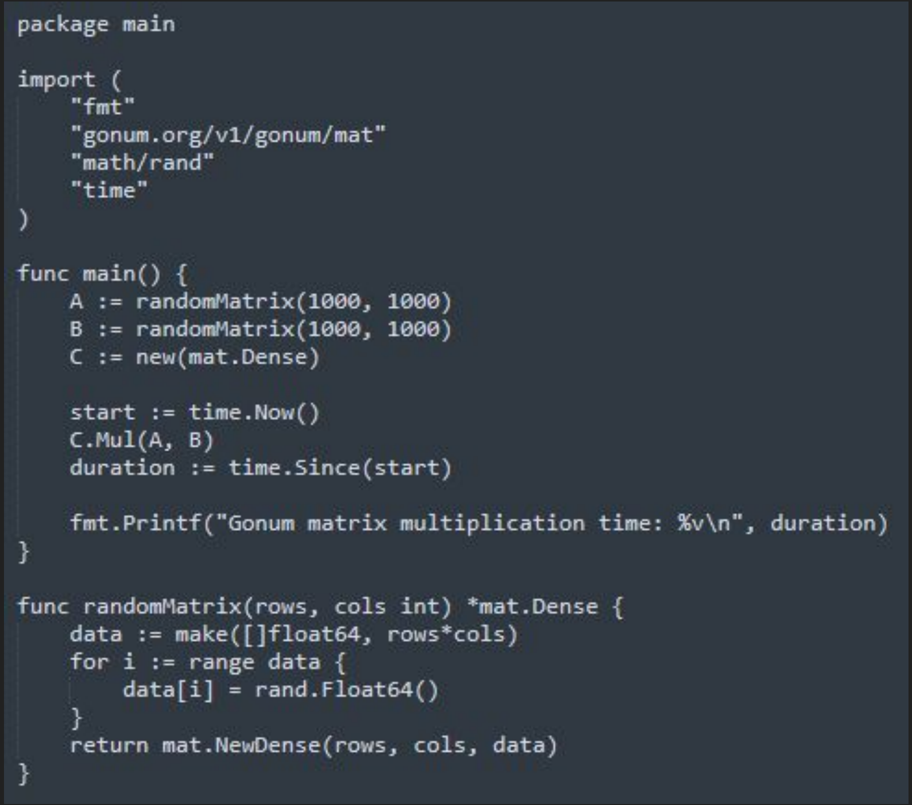


```go
package main

import (
    "fmt"
    "gonum.org/v1/gonum/mat"
)

func main() {
    A := mat.NewDense(3, 2, []float64{1, 2, 3, 4, 5, 6})
    B := mat.NewDense(2, 3, []float64{7, 8, 9, 10, 11, 12})
    C := new(mat.Dense)
    C.Mul(A, B)

    fmt.Println("Matrix A:")
    matPrint(A)

    fmt.Println("Matrix B:")
    matPrint(B)

    fmt.Println("Matrix C:")
    matPrint(C)
}

func matPrint(X mat.Matrix) {
    formattedMatrix := mat.Formatted(X, mat.Prefix(""), mat.Excerpt(0))
    fmt.Printf("%v\n\n", formattedMatrix)
}
```

Researching Benchmarking Techniques

For research purposes, I have attempted to run benchmarking on the matrix multiplication function for GoNum, Julia and NumPy.

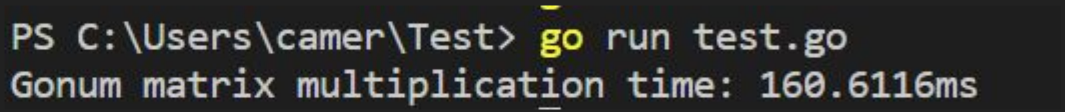
The following is my code and results.



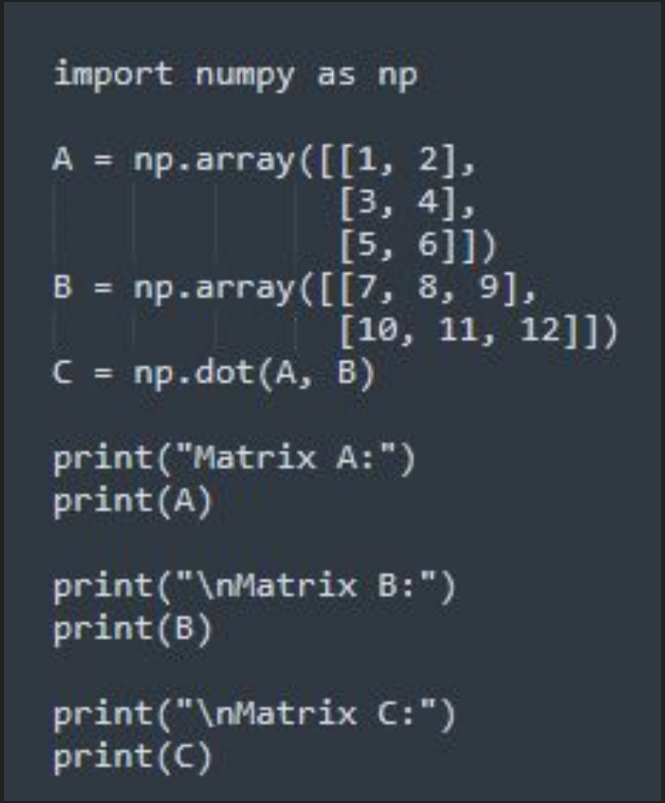
```go
package main

import (
	"fmt"
	"gonum.org/v1/gonum/mat"
	"math/rand"
	"time"
)

func main() {
	A := randomMatrix(1000, 1000)
	B := randomMatrix(1000, 1000)
	C := new(mat.Dense)

	start := time.Now()
	C.Mul(A, B)
	duration := time.Since(start)

	fmt.Printf("Gonum matrix multiplication time: %v\n", duration)
}

func randomMatrix(rows, cols int) *mat.Dense {
	data := make([]float64, rows*cols)
	for i := range data {
		data[i] = rand.Float64()
	}
	return mat.NewDense(rows, cols, data)
}
```

Results from Benchmarking GoNum Matrix Multiplication

The results show that Go Matrix Multiplication is pretty quick. However, my interest is to compare this time and the timing of other functions with that of NumPy and Julia. I hope to investigate this further given more time!

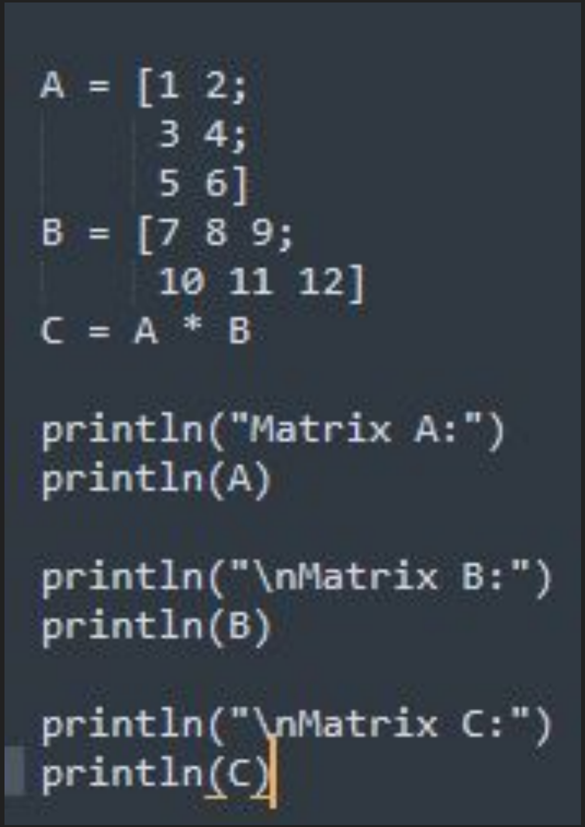


```
PS C:\Users\camer\Test> go run test.go
Gonum matrix multiplication time: 160.6116ms
```



```python
import numpy as np

A = np.array([[1, 2],
              [3, 4],
              [5, 6]])
B = np.array([[7, 8, 9],
              [10, 11, 12]])
C = np.dot(A, B)

print("Matrix A:")
print(A)

print("\nMatrix B:")
print(B)

print("\nMatrix C:")
print(C)
```

NumPy



```julia
A = [1 2;
     3 4;
     5 6]
B = [7 8 9;
     10 11 12]
C = A * B

println("Matrix A:")
println(A)

println("\nMatrix B:")
println(B)

println("\nMatrix C:")
println(C)
```

Julia

Lingering Questions

What limitations does GoNum have when compared to a fully fledged package like NumPy?

Is Go truly anymore efficient that its counterparts when doing complex data manipulation using linear algebra functions?

Does the GoNum library fall short of its counterparts in terms of offerings?

Proposed Experiment

For my project, I hope to perform benchmark testing on various functions in Go's, Gonum Matrix Library. I hope to compare this benchmark to one's I will carry out on NumPy and Julia. Alongside benchmarking of these libraries' linear algebra functions I hope to compare the accuracy of these functions as well. I have been interested in whether or not Go's reputation for efficiency will be applicable when completing data-intensive tasks involving numerical computations. Also if Go *is* more efficient, it will be all the more interesting to compare accuracy of the results.