RAG 2015 final report

Ruby Association Grant 2015 Final Report

Author: Sameer Deshmukh

Email : sameer.deshmukh93@gmail.com

GitHub: @v0dro

Contents

Overview

The NMatrix linear algebra library has been gaining popularity steadily over the past few years. It received a major overhaul by Will Levine in 2015 as part of that year's Google Summer of Code. He implemented a very helpful plugin architecture for NMatrix which allows programmers to take advantage of the data structures and C data types of NMatrix and interface them with any C library for performing fast computations. Examples of such plugins are nmatrix-lapacke and nmatrix-atlas.

This project takes advatage of these recent developments in NMatrix and improves the functionality of NMatrix by creating better interfaces to two widely used numerical libraries — FFTW and GSL. The interfaces have been created by leveraging the C-level API that NMatrix provides, and thus performing most of the performance intensive tasks in native code, while exposing a friendly Ruby interface to the user.

This has culminated in a new plugin for nmatrix called **nmatrix-fftw** for creating a Ruby wrapper using NMatrix over FFTW, and NMatrix support for the GNU Scientific Library through the rb-gsl gem.

Results

nmatrix-fftw

Before starting this project, I did some research to see if any FFTW interfaces for Ruby had already been built. My search lead to the fftw library by Magdalen Berns. The biggest problem with Magdalen's library was that it did not leverage FFTW's or NMatrix's unique API, and it was also very old and not being maintained. The code base was still pretty usable so I decided to use it as a starting point and make my NMatrix FFTW plugin using the latest NMatrix architecture and also design a better Ruby API.

This led to creation of the nmatrix-fftw library, that is basically a Ruby wrapper over the FFTW C library. It takes advantage of FFTW's plans and gives the user complete control of his FFT computation, all using NMatrix as a very convenient data store.

A Fast Fourier Transform can be created and executed as follows:

```
require 'nmatrix'
require 'nmatrix/fftw'

input = NMatrix.new([10],
  [
    Complex(9.32,0), Complex(44,0), Complex(125,0), Complex(34,0),
    Complex(31,0),   Complex(44,0), Complex(12,0),  Complex(1,0),
    Complex(53.23,0),Complex(-23.23,0),
  ], dtype: :complex128)
plan = NMatrix::FFTW::Plan.new(10)
plan.set_input input
```

```
plan.execute
print plan.output
# =>
#[(330.32+0.0i), (-8.403943680631363-150.3269135174232i), (-99.48067997927964-68.65789001229444i),
#(-143.68605631936865-20.427342517448675i), (67.62067997927964+8.523578418697124i), (130.78+0.0i),
#(67.62067997927964-8.523578418697124i), (-143.68605631936865+20.427342517448675i),
#(-99.48067997927964+68.65789001229444i), (-8.403943680631363+150.3269135174232i)]
```

The `NMatrix::FFTW::Plan.new()` method supports many options which let a user modify the manner in which the FFT will be eventually computed. A complete list can be found in the docs.

This was the work done until the mid term review period.

You can see the code here, and the tests here. The gem has been released on rubygems.

Ruby/GSL

The GNU Scientific Library (GSL) is a very robust open source library that contains hundreds of useful routines for computation of everything from matrix factorizations to wavelet transforms. The rb-gsl Ruby gem is a Ruby wrapper over GSL. It previously accepted NArray and GSL data types and supported interconversion between them (it still does). NArray is not being very actively maintained anymore, and given the rising attention that NMatrix has been receiving, it makes sense to have NMatrix compatibility built into rb-gsl so that NMatrix users can leverage the vast number of routines provided by GSL.

Making NMatrix compatible with rb-gsl and upgrading rb-gsl to support GSL v2.1 (it previously supported upto v1.16 only) constitutes my final term work.

I first started off with making sure that NMatrix and GSL data types are able to convert between each other, i.e. interconversions between `GSL::Vector` / `GSL::Matrix` and `NMatrix` are now possible. I started with the `#to_nm` function for GSL::Vector and GSL::Matrix for converting them to NMatrix:

```
require 'gsl'

v = GSL::Vector.alloc(1,2,3,4)
# => GSL::Vector
# [ 1.000e+00 2.000e+00 3.000e+00 4.000e+00 ]
v.to_nm
# => [1.0, 2.0, 3.0, 4.0]
m = GSL::Matrix[[1,2],[3,4]]
# => GSL::Matrix
# [  1.000e+00  2.000e+00
#    3.000e+00  4.000e+00 ]
m.to_nm
# =>
# [
#   [1.0, 2.0]   [3.0, 4.0] ]
```

These methods work as per the data type of the GSL container (`int` , `double` or `complex`) and produce the corresponding NMatrix. The reverse is also possible. `GSL::Matrix` can be directly converted to a 2D NMatrix using the `NMatrix#to_gslm` method and `GSL::Vector` to a 1D Vector NMatrix using the `NMatrix#to_gslv` method. These methods will reflect the data type of the data in the NMatrix that they create. Thus an NMatrix with dtype `:int32` will produce a `GSL::Matrix::Int` .

```
require 'gsl'
vec = NMatrix.new([5], [1,2,3,4,5], dtype: :int32)
# => [1, 2, 3, 4, 5]
vec.to_gslv
# => GSL::Vector::Int
# [ 1 2 3 4 5 ]
mat = NMatrix.new([2,2], [4,56,2,1])
# =>
# [
#   [4, 56]   [2,  1] ]
mat.to_gslm
# => GSL::Matrix::Int
# [  4 56
#    2  1 ]
```

After finishing interconversions, I made changes to various rb-gsl methods so that they can accept NMatrix objects without any major efforts on part of the user. A list of methods that can accept NMatrix objects can be found here. I also made sure that rb-gsl works properly with the latest version of GSL (v2.1), and thus rb-gsl will be ready to work with the latest versions of GSL that will be included in the upcoming Debian release.

The final review work is completely is done, and is pending approval from the other maintainers of rb-gsl. The Pull Request can be found here. Once done, a new version of rb-gsl will be released on rubygems.

Future Work

As far as nmatrix-fftw is concerned, most of the methods that are made available by FFTW have been exposed through the Ruby wrapper. A few specialized methods remain to be exposed, but they are not very critical or widely used, so they will be made available as and when users file issues.

In case of rb-gsl, more work will involve making all the rb-gsl methods work with nmatrix and also exposing all of GSL's functionality through the Ruby interface.

Conclusion

I conclude that all the work that I had promised in the grant period has been accomplished successfully and has laid the ground work for much greater improvements to the Ruby scientific stack. Improvements to the existing code will be done when many users start using these libraries and provide us with their valuable feedback.

I would like to specially thank the maintainers at the Ruby Science Foundation for their contributions in making all this happen.