

# Differentiating sparse matrix operations with reversible programming

Jie Li

Mentors: Jinguo Liu, Jiuning Chen

September 30, 2021

## 1 Project Information

### 1.1 Scheme Description

Sparse matrices are extensively used in scientific computing, however there is no automatic differentiation package in Julia yet to handle sparse matrix operations. This project utilizes the reversible embedded domain-specific language NiLang.jl to differentiate sparse matrix operations by writing the sparse matrix operations in a reversible style. The generated backward rules are ported to ChainRules.jl as an extension, so that one can access these features in an automatic differentiation package like Zygote, Flux and Diffactor directly.

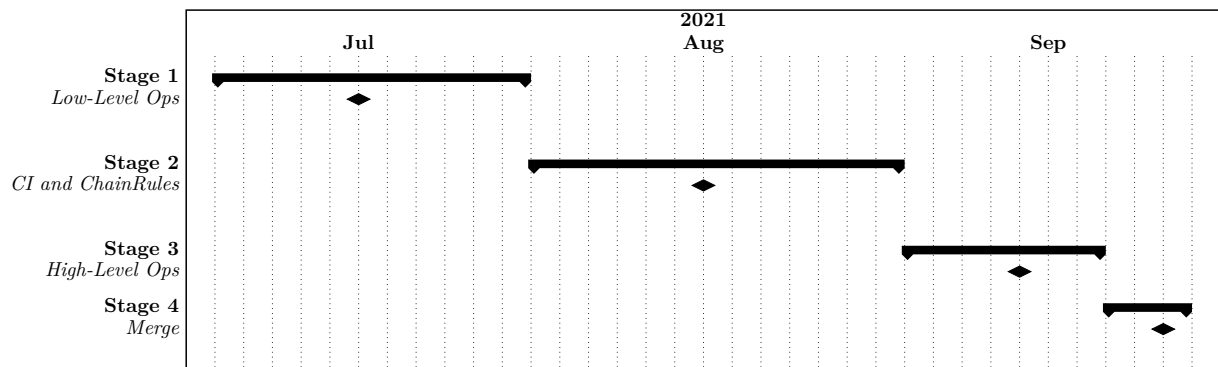
### 1.2 Time Planning

This project is shipped by four (mostly) sequential stages:

- (1) Implement low level operations by NiLang.
- (2) Carefully test by CI and export chain rules into ChainRules.jl.
- (3) Implement high level operations and perform AD operations.
- (4) Add some use cases and enhance docs.

I list the timeline of this project in the form of Gantt chart.

- (1) 1st Jul - 31st Jul Implement low level operations by NiLang.
- (2) 1st Aug - 15th Aug Carefully test by CI and export chain rules into ChainRules.jl.
- (3) 15th Sep - 20th Sep Implement high level operations and perform AD operations.
- (4) 21st Sep - 30th Sep Add some use cases and enhance docs.



## 2 Project Summary

### 2.1 Project Output

- (1) Differentiate sparse matrix operations in Julia base by rewriting the sparse functions in NiLang.jl.  
**Status:** *Already completed. Black-box tests on NiLang programs have passed in CI.* Related PR
- (2) Port the generated backward rules to ChainRules.jl as an extension.  
**Status:** *Already completed. Backward rules have been checked by ChainRulesTestUtils.* Related PR
- (3) Release an open source julia package with test coverage over 85%.  
**Status:** *Test coverage achieved at 87%.* Code Converage Report
- (4) Add some use cases for getting a start.  
**Status:** *Already completed. A simple use case is shown in ReadMe.* Related PR

### 2.2 Scheme Progress

1. 1st Jul - 31st Jul Implement low level operations by NiLang.  
*I have rewritten almost all sparse matrix multiplication and dot operations in Julia base by NiLang. I list the implemented sparse matrix operators as follows. All the functions listed have passed CI tests and the code coverage have achieved over 85%.*

```
1      function imul!(C::StridedVecOrMat, A::AbstractSparseMatrix{T}, B::DenseInputVecOrMat, ::  
      Number, ::Number) where T  
2      function imul!(C::StridedVecOrMat, xA::Adjoint{T, <:AbstractSparseMatrix}, B::  
      DenseInputVecOrMat, ::Number, ::Number) where T  
3      function imul!(C::StridedVecOrMat, X::DenseMatrixUnion, A::AbstractSparseMatrix{T}, ::Number,  
      ::Number) where T  
4      function imul!(C::StridedVecOrMat, X::Adjoint{T1, <:DenseMatrixUnion}, A::  
      AbstractSparseMatrix{T2}, ::Number, ::Number) where {T1, T2}  
5      function imul!(C::StridedVecOrMat, X::DenseMatrixUnion, xA::Adjoint{T, <:AbstractSparseMatrix  
      }, ::Number, ::Number) where T  
6      function idot(r, A::SparseMatrixCSC{T}, B::SparseMatrixCSC{T}) where {T}  
7      function idot(r, x::AbstractVector, A::AbstractSparseMatrix{T1}, y::AbstractVector{T2}) where {  
      T1, T2}  
8      function idot(r, x::SparseVector, A::AbstractSparseMatrix{T1}, y::SparseVector{T2}) where {T1,  
      T2}  
9
```

2. 1st Aug - 15th Aug Carefully test by CI and export chain rules into ChainRules.jl.  
*I have exported the backward rules into ChainRules.jl. I used ChainRulesTestUtils.jl to test correctness and robustness of sparse matrix AD rules. All the rules have passed CI tests.*
3. 15th Sep - 20th Sep Implement high level operations and perform AD operation.  
*I have implemented low rank svd algorithm in Julia. To perform AD on low\_rank\_svd algorithm, I wrapped backward rules of QR decomposition since QR rules haven't been ensembled in ChainRules. I used finite differences method to check the correctness of the algorithm. All the code have passed CI tests and the code converage have achieved at 87% finally.*
4. 21st Sep - 30th Sep Add some use cases and enhance docs.  
*A simple use case has been added into ReadMe in the project.*

### 2.3 Problems and Solutions

To complete the projcet, one should be equipped with both good sense of numerical linear algebra and automatic differentiation **in theory** and familiarity with NiLang.jl, ChainRules.jl and so on **in practice**. It took me much of time to figure out how automatic differentiation works out in computation process. In this section, I would only introduce the problems and solutions I came cross in practice.

### 2.3.1 Incorrect gradient of sparse vector

When I tried to export chain rules of function *dot*, all the checks failed in `ChainRulesTestUtils`. Then, I printed the gradient calculated by `NiLang` and found it proved to be zero which was really abnormal. I fed back the strange phenomenon to mentor and he told me that iterator can not carry gradients and the dangerous feature would be removed from `NiLang` in the future. After I corrected the codes according to his suggestions, all the checks passed. The problem also gave a good reason to remove iterator support to avoid potential missing of gradients in `NiLang`, which had been mentioned in this issue.

### 2.3.2 Test on Generated Rules

In our project, I calculated the gradient of sparse matrix in reversal mode in `NiLang`. To test the correctness of the gradient, I should calculate gradient in another way. There are many choices such as `FiniteDifferences`, `ForwardDiff` and so on. After discussing with mentors, I decided to use `ChainRulesTestUtils` to check the correctness of the gradient, which was highly recommended by Lyndon White in `JuliaCon2020`. At first, I implemented tangent of sparse matrix by myself, it is a bit hard task for me since there are no related examples. Later, mentor and I talked about this problem with Lyndon White, the core founder of `ChainRules`, in slack and he mentioned `rand_tangent` method may help. I read related source code in `CRTU` and made some changes so as to keep the adjoint gradient could keep the sparsity. Finally all the rules checked by `CRTU` and passed all the tests successfully.

### 2.3.3 Add support for QR rules

The `low_rank_svd` involved with QR decomposition which was used to find the approximate basis. However, backward rules of QR decomposition were ensembled neither in `ChainRules` nor `Zygote`. In order to calculate the gradient of `low_rank_svd`, I wrapped the QR rules in `ChainRules` and checked by `CRTU` in help of mentor. I would like to enhance the implementation of QR rules (keep Q factor and R factor in compactWY format) and pull request to `ChainRules.jl` in the future.

## 2.4 Development Quality

You could give a self evaluation of development quality

## 2.5 Communication and Feedback with Mentor