Implementing the Truncated Density Convolution

Kangbo Li

March 13, 2024

1 Theory Overview

1.1 The objective function

This document describes the details of what we want to implement. The optimization problem can be written in abstract as

$$\min_{U} \ \Omega(U), \tag{1}$$

$$\text{s.t } U^{k\dagger}U^{k} = I \in \mathbb{C}^{N_e \times N_e}, \tag{2}$$

$$\Omega(U) = 2\sum_{n,b} w_b(1 - |\hat{\rho}_n(b)|), \quad \hat{\rho}_n(b) = \frac{1}{N_k}\sum_{k}(U^{k\dagger}S^{k,b}U^{\mathrm{add}(k,b)})_{n,n}, \tag{3}$$

$$U \in \mathbb{C}^{N_j \times N_e \times N_k}, S \in \mathbb{C}^{N_j \times N_j \times N_k \times N_b}, w \in \mathbb{R}^{N_b}. \tag{4}$$

The variables represents

1. N_k: the number of k-points.
2. N_b: the number of b-vectors.
3. N_e: the number of electrons per unit cell.
4. N_j: the number of bands to disentangle. $N_j = N_e$ for insulators.
5. w_b: the weights corresponding to each b vector.
6. U: the gauge.
7. S: the overlap/transition matrices.
8. Ω: the total spread.
9. $\hat{\rho}_n$: the Fourier transform of the density of the n^{th} electron.

This is a constrained minimization problem, where the independent variable is a complex tensor. The gradient of the objective function is

$$(\nabla\Omega(U))^{k}_{ij} = -\frac{4}{N_k}\sum_{b,s} w_b \frac{\hat{\rho}^{*}_{j}(b)}{|\hat{\rho}_j(b)|} S^{k,b}_{i,s} U^{\mathrm{add}(k,b)}_{s,j}. \tag{5}$$

The gist of the project is to make Ω and $\nabla\Omega$ fast. The add function is a world of hurt, so we will just tabulate the result of the reference code and not worry about it.

1.2 Manifold Optimization

This constrained optimization problem is solved with a manifold conjugate gradient descent. The algorithm for a nonlinear conjugate gradient is roughly

```
function cg(f, grad_f)
    x = zeros
    p = grad_f(x)
    while not converged
        x = x - alpha * p
        p = grad_f(x) + beta * p
    end
end
```

The step length `alpha` is determined by a (quadratic) line search whereas the formula for `beta` has many variants. The one implemented in the reference code is the Fletcher Reeves.

It would be ideal if we could use an off-the-shelf optimizer, but the complication is that we are working on a manifold, which modifies the algorithm

```
function cg(f, grad_f)
    x = zeros
    p = project(grad_f(x))
    while not converged
        x = retract(x, -alpha * p)
        p = project(grad_f(x)) + beta * p
    end
end
```

For the Stiefel manifold, the projection and retraction are

$$\mathrm{project}(C, X) = X - C\left(\frac{C^\dagger X + X^\dagger C}{2}\right), \tag{6}$$

$$\mathrm{retract}(U, \Delta U) = \mathrm{ortho}(U + \Delta U). \tag{7}$$

These formula can be simplified a bit in the special case of the orthogonal group, which corresponds to the insulators. The inefficient orthogonalization that leads to the algorithm in the MLWF literature is the exponential map. The simplified projection and the exponential map are

$$\mathrm{project}(U, X) = U\left(\frac{U^\dagger X - X^\dagger U}{2}\right), \tag{8}$$

$$\mathrm{retract}(U, \Delta U) = U\exp\bigl(U^\dagger \Delta U\bigr). \tag{9}$$

To save some flops, one can move one of the Us from the projection to the retraction because the retraction step typically follows the projection in manifold optimization.

$$\mathrm{project}(U, X) = (U^\dagger X - X^\dagger U)/2, \tag{10}$$

$$\mathrm{retract}(U, \Delta U) = U\exp(\Delta U). \tag{11}$$

In terms of computational cost, one ends up with two matrix multiplications and a matrix exponential. This is no cheaper than the Stiefel projection/retraction that we started with with the additional limitation to the orthogonal group.

2 Implementation Plan

1. (x) Prepare the parameters of the problem.

2. (x) Read the parameters from Fortran.
3. (x) Serial implementation of the objective function, the gradient, and the retraction.
4. (x) Implement a manifold optimization.
5. Implement disentanglement.
6. Speedup.

2.1 Prepare the data

The S and w are written in a Fortran binary file just as raw Fortran arrays whose dimensions match Eq. 4. The **add** function is tabulated and saved to a Fortran binary file as a $N_k \times N_b$ array, where the k^{th} row and b^{th} column is the new index $\mathrm{add}(k, b)$.