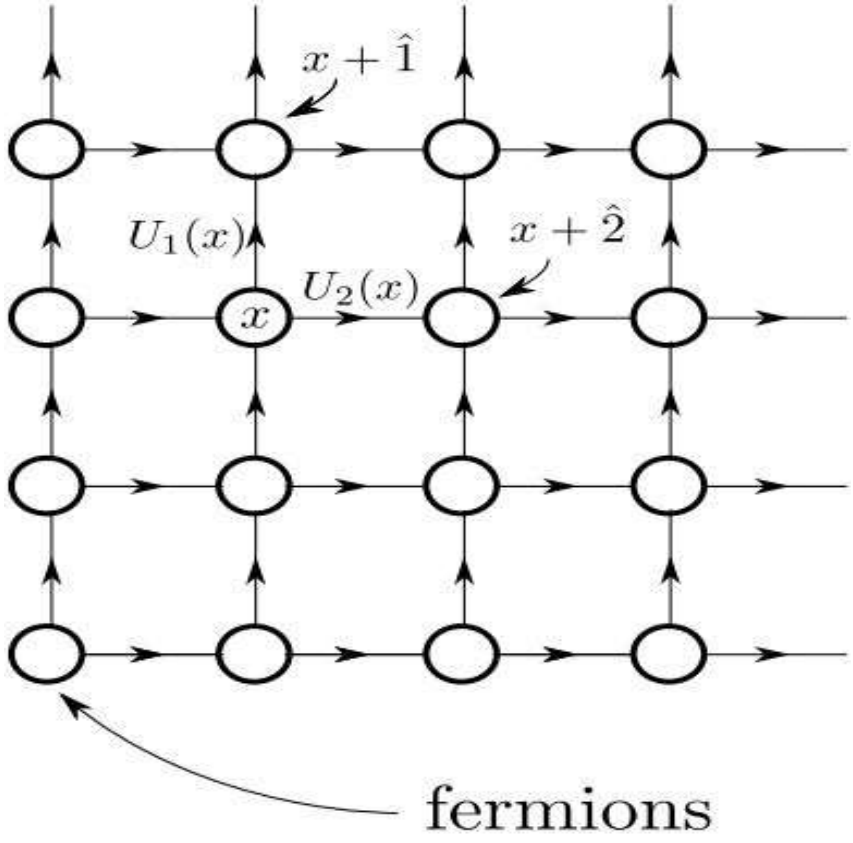
The Inversion of a Dirac Matrix

A Raihan Bhuiyan 2130846
Md Imran Nur 1942523
Hamid Mohammadpour 2130766

Supervised by: Dr. Lukas Varnhorst



Winter semester 2022-2023

Task a: Construct a neighbor table, i.e. a 4 × (N1N2) dimension array nμx. Here μ correspond

the four possible directions (forward in directions 1 or 2 or backwards in direction 1 or 2) and x is th index of a lattice point. Then nμx should contain the index of the neighbor of the site with index x ir direction.

```
for(y=0; y<g_N1; y++){
    for(x=0; x<g_N2; x++){
        Cell c (
            x,
            y,
            x+y*g_N1,
            lex_to_top_lex(x, y),
            f_top_lex_antiperiodic(x, y),
            lex_to_bottom_lex(x, y),
            f_bottom_lex_antiperiodic(x, y),
            lex_to_right_lex(x, y),
            f_right_lex_antiperiodic(x, y),
            lex_to_left_lex(x, y),
            f_left_lex_antiperiodic(x, y)
        );

        lattice.push_back(
                c
            );

    }
}
```

Two for loops used to construct the lattice

x: 0 y: 0 lex: 0 top (lex): 4 bottom (lex): 12 right (lex): 1 left (lex): 3 U_mu: 0 + 0i	x: 1 y: 0 lex: 1 top (lex): 5 bottom (lex): 13 right (lex): 2 left (lex): 0 U_mu: 0 + 0i	x: 2 y: 0 lex: 2 top (lex): 6 bottom (lex): 14 right (lex): 3 left (lex): 1 U_mu: 0 + 0i	x: 3 y: 0 lex: 3 top (lex): 7 bottom (lex): 15 right (lex): 0 left (lex): 2 U_mu: 0 + 0i
x: 0 y: 1 lex: 4 top (lex): 8 bottom (lex): 0 right (lex): 5 left (lex): 7 U_mu: 0 + 0i	x: 1 y: 1 lex: 5 top (lex): 9 bottom (lex): 1 right (lex): 6 left (lex): 4 U_mu: 0 + 0i	x: 2 y: 1 lex: 6 top (lex): 10 bottom (lex): 2 right (lex): 7 left (lex): 5 U_mu: 0 + 0i	x: 3 y: 1 lex: 7 top (lex): 11 bottom (lex): 3 right (lex): 4 left (lex): 6 U_mu: 0 + 0i
x: 0 y: 2 lex: 8 top (lex): 12 bottom (lex): 4 right (lex): 9 left (lex): 11 U_mu: 0 + 0i	x: 1 y: 2 lex: 9 top (lex): 13 bottom (lex): 5 right (lex): 10 left (lex): 8 U_mu: 0 + 0i	x: 2 y: 2 lex: 10 top (lex): 14 bottom (lex): 6 right (lex): 11 left (lex): 9 U_mu: 0 + 0i	x: 3 y: 2 lex: 11 top (lex): 15 bottom (lex): 7 right (lex): 8 left (lex): 10 U_mu: 0 + 0i
x: 0 y: 3 lex: 12 top (lex): 0 bottom (lex): 8 right (lex): 13 left (lex): 15 U_mu: 0 + 0i	x: 1 y: 3 lex: 13 top (lex): 1 bottom (lex): 9 right (lex): 14 left (lex): 12 U_mu: 0 + 0i	x: 2 y: 3 lex: 14 top (lex): 2 bottom (lex): 10 right (lex): 15 left (lex): 13 U_mu: 0 + 0i	x: 3 y: 3 lex: 15 top (lex): 3 bottom (lex): 11 right (lex): 12 left (lex): 14 U_mu: 0 + 0i

All the Points of the Lattice

Task b: Using the result of a), write a function that applies the above matrix (with Uμ (x) = 1 f

simplicity and m = 0.4) to a arbitrary vector v ∈ C^N1N2 , i.e. that calculates Dv. Note that the funct

should not explicitly construct D

```
double D_val(int point_x_lex, int point_y_lex, const vector<Cell>& lattice, bool conjugate_transpose=false){
    // variable
    double m = 0.4;
    // conditional delta
    double delta_x_y{0.0};
    double delta_x_r_y{0.0};
    double delta_x_l_y{0.0};
    double delta_x_t_y{0.0};
    double delta_x_b_y{0.0};
    double result{0.0};

    // it mentioned that U_n(x) = 1 for simplicity
    double U1_x{1.00};
    double U2_x{1.00};
    double U1_star_x_l{1.00};
    double U2_star_x_b{1.00};

    // check for delta_x_y
    if (point_x_lex == point_y_lex){

    // check for delta_x_r_y
    if (lattice[point_x_lex].right_lex == point_y_lex){

    // check for delta_x_l_y
    if (lattice[point_x_lex].left_lex == point_y_lex){

    // check for delta_x_t_y
    if (lattice[point_x_lex].top_lex == point_y_lex){

    // check for delta_x_b_y
    if (lattice[point_x_lex].bottom_lex == point_y_lex){

    result = m * delta_x_y +
                (U1_x * delta_x_r_y - U1_star_x_l * delta_x_l_y) / 2.0 +
                pow((-1), lattice[point_x_lex].x) * (U2_x * delta_x_t_y - U2_star_x_b * delta_x_b_y) / 2.0;

    return result;
}
```

The function to construct D

Task b: Using the result of a), write a function that applies the above matrix (with Uμ (x) = 1 f

simplicity and m = 0.4) to a arbitrary vector v ∈ C^N1N2 , i.e. that calculates Dv. Note that the funct

should not explicitly construct D

```
vector<double> mv_D_v(int g_DN1, int g_DN2, vector<double> v, vector<Cell> lattice){
    vector<double> new_v(g_DN1, 0.0);
    int i, j;
    double d_x_y;

    // calculate D* dot v
    for(i=0; i<g_DN1; i++){
        for(j=0; j<g_DN1; j++){
            if(v[j] != 0.0){
                d_x_y = D_val(i, j, lattice, true)*v[j];
            }
        }
        new_v[i] = d_x_y;
        // cout << "idx: " << i << ", value=" << d_x_y << endl;
    }

    return new_v;
}
```

The function requested in task b

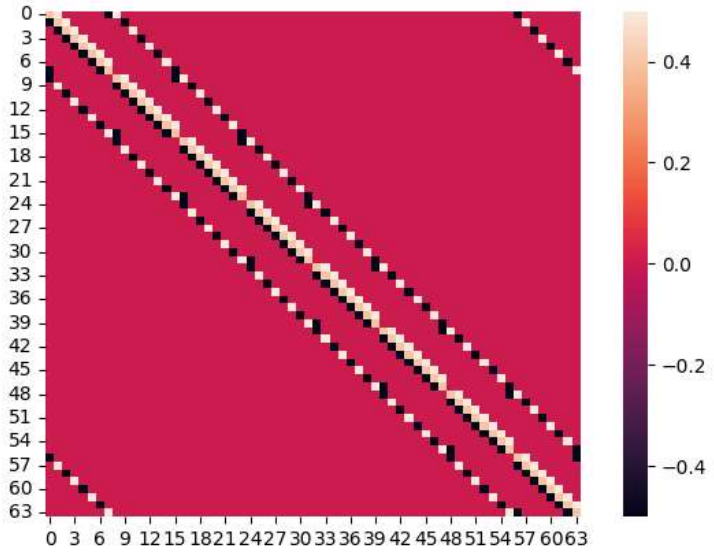
Task c: For a small 8 × 8 lattice, apply the function from b) to each of the unit vectors to cons

the full matrix. Plot the magnitude of the matrix elements in a 2d plot, so that the structure of the m
can be observed

```
Printing result ...
-2.16493e-13
0.431034
1.65257e-13
2.45137e-13
-0.431034
0.344828
0.431034
9.05942e-14
-6.31162e-14
-0.431034
7.31672e-14
-2.15605e-13
3.16192e-13
1.31228e-13
-2.23321e-13
-2.13163e-13
```

Task c: For a small 8 × 8 lattice, apply the function from b) to each of the unit vectors to cons

the full matrix. Plot the magnitude of the matrix elements in a 2d plot, so that the structure of the m
can be observed



The constructed D Matrix

Task d: Implement the CG algorithm discussed in the lecture and use it to calculate D^−1ei w

ei is the ith unit vector. Note that D is not symmetric and positive definite, so you have to solve the equation (D†D)r = D†ei for r. Construct the matrix D −1 from the result and plot it like in c)

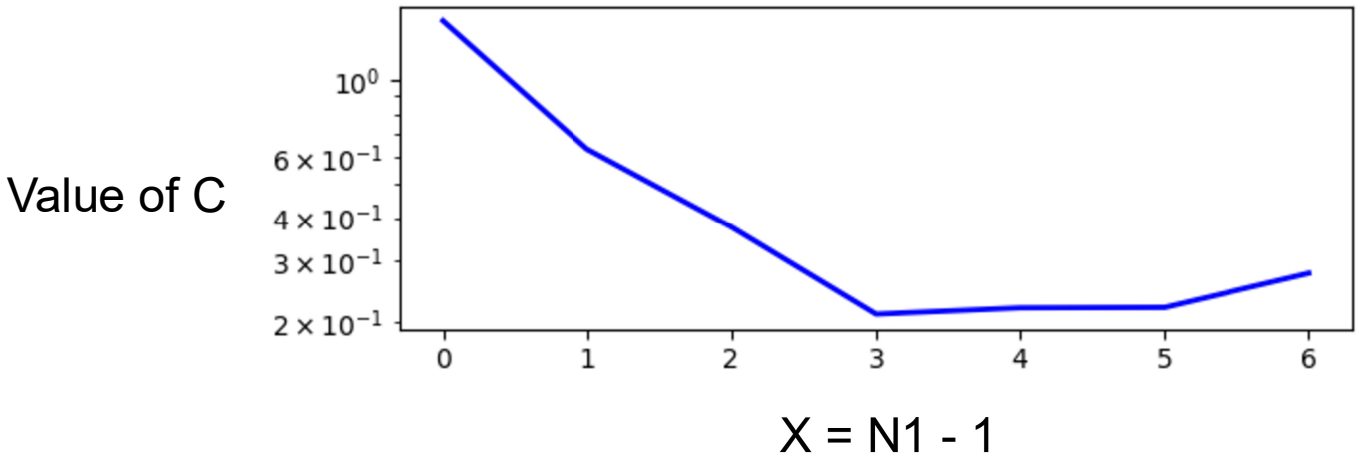
```
vector<double> LinearCG(vector<double> A, vector<double> b, vector<double> x0, int A_row, int A_col, int x_col){···

vector<double> LinearCG_nonSPD(vector<double> b, vector<double> x0, vector<Cell> lattice, int A_row, int A_col, int x_col){···
```

The function used to calculate task d

Task e: Implementation of Correlation Function

```
double d_inv_e_i_term{0.0};
for(t=0; t<g_N2-1; t++){
    for(j=0; j<g_N1; j++){
        for(i=0; i<g_N2; i++){
            xk = inD_e_i[i + (j-1) * g_N2];
            d_inv_e_i_term += xk[((i + (j-1) * g_N2)+t)%g_DN1]*xk[((i + (j-1) * g_N2)+t)%g_DN1];
        }
    }
    ofile_c_pointer << d_inv_e_i_term/((double)g_N1*g_N2) << ",";
    d_inv_e_i_term = 0.0;
}
```

Task e: Implementation of Correlation Function



Task f: Read the given input of D

```
ifstream D_example_file("example_for_U.dat.txt");
// i=>column or x, j=>row or y, x=>direction, d_x_y is the value of U
vector<Cell*> D_cells;
while (D_example_file >> i >> j >> x >> d_x_y){
    // cout << i << ", " << j << ", "<< x << ", " << d_x_y << endl;
    D_cells.push_back(
                new Cell(
                    i, j, x, d_x_y
                )
            );
}
```

Task h: Parallelize of N1XN2 matrix

lex 0, x=0, y=0	lex 1, x=1, y=0	lex 2, x=2, y=0	lex 3, x=3, y=0	lex 4, x=4, y=0	lex 5, x=5, y=0	lex 6, x=6, y=0	lex 7, x=7, y=0
lex 8, x=0, y=1	lex 9, x=1, y=1	lex 10, x=2, y=1	lex 11, x=3, y=1	lex 12, x=4, y=1	lex 13, x=5, y=1	lex 14, x=6, y=1	lex 15, x=7, y=1
lex 16, x=0, y=2	lex 17, x=1, y=2	lex 18, x=2, y=2	lex 19, x=3, y=2	lex 20, x=4, y=2	lex 21, x=5, y=2	lex 22, x=6, y=2	lex 23, x=7, y=2
lex 24, x=0, y=3	lex 25, x=1, y=3	lex 26, x=2, y=3	lex 27, x=3, y=3	lex 28, x=4, y=3	lex 29, x=5, y=3	lex 30, x=6, y=3	lex 31, x=7, y=3
lex 32, x=0, y=4	lex 33, x=1, y=4	lex 34, x=2, y=4	lex 35, x=3, y=4	lex 36, x=4, y=4	lex 37, x=5, y=4	lex 38, x=6, y=4	lex 39, x=7, y=4
lex 40, x=0, y=5	lex 41, x=1, y=5	lex 42, x=2, y=5	lex 43, x=3, y=5	lex 44, x=4, y=5	lex 45, x=5, y=5	lex 46, x=6, y=5	lex 47, x=7, y=5
lex 48, x=0, y=6	lex 49, x=1, y=6	lex 50, x=2, y=6	lex 51, x=3, y=6	lex 52, x=4, y=6	lex 53, x=5, y=6	lex 54, x=6, y=6	lex 55, x=7, y=6
lex 56, x=0, y=7	lex 57, x=1, y=7	lex 58, x=2, y=7	lex 59, x=3, y=7	lex 60, x=4, y=7	lex 61, x=5, y=7	lex 62, x=6, y=7	lex 63, x=7, y=7

Task h: Parallelize of N1XN2 matrix

		lex 60, x=4, y=7	lex 61, x=5, y=7	lex 62, x=6, y=7	lex 63, x=7, y=7		
	lex 3, x=3, y=0	lex 4, x=4, y=0	lex 5, x=5, y=0	lex 6, x=6, y=0	lex 7, x=7, y=0	lex 0, x=0, y=0	
	lex 11, x=3, y=1	lex 12, x=4, y=1	lex 13, x=5, y=1	lex 14, x=6, y=1	lex 15, x=7, y=1	lex 8, x=0, y=1	
	lex 19, x=3, y=2	lex 20, x=4, y=2	lex 21, x=5, y=2	lex 22, x=6, y=2	lex 23, x=7, y=2	lex 16, x=0, y=2	
	lex 27, x=3, y=3	lex 28, x=4, y=3	lex 29, x=5, y=3	lex 30, x=6, y=3	lex 31, x=7, y=3	lex 24, x=0, y=3	
		lex 36, x=4, y=4	lex 37, x=5, y=4	lex 38, x=6, y=4	lex 39, x=7, y=4		

p=3 (4 processor in total), local matrix: 4X4, 8X8 matrix in total

Task i: Generation of D using only local points