Computational Physics III: Report 3
Steepest descent algorithm and singular value decomposition

Due on Mai 30nd, 2019

Viktor Crettenand

Contents

Problem 1 **3**
Steepest descent and conjugate gradient methods 3
1 3
2 4
3 4
4 5
Problem 2 **6**
Problem 3 **7**
Problem 4 **8**
Problem 5 **11**

Problem 1

Steepest descent and conjugate gradient methods

1

In this exercise puzzles are solved. The puzzles is a board of size $N \times N$ with a positive integer or zero number of coins in each cell. At each step the player can select a cell and remove a coin from it and all its edge sharing neighbours. Solving a puzzle is equivalent to solving the following system of linear equations (here given in matrix form):

$$b = A \cdot x \tag{1}$$

b is a column vector containing the number of coins in each cell. The first N elements of column vector b are the number of coins in the cells of the puzzle's first row, the following N next are the number of coins in the cells of the puzzle's second row and so on. A is the problems matrix whose form is not justified here but simply given: $A = T \otimes \mathbb{1} + \mathbb{1} \otimes T - \mathbb{1} \otimes \mathbb{1}$. Finally x is the solution of the problem, i.e. a vector of the same form and with the same convention as vector b with it's components being the number of times each cell needs to be selected by the player. The order the cells are selected in doesn't matter.

A function named "puzzleA" was implemented to generate a matrix A of size $N \times N$ and the following example puzzle was solved:

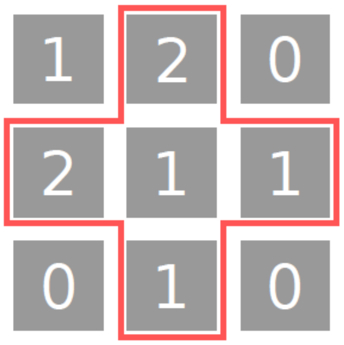


Figure 1: Example puzzle

The result found is:

$$x = \begin{bmatrix} 1 \\ 0 \\ 0 \\ 0 \\ 1 \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix} \tag{2}$$

A function called "matrixform" was created to put this vector in a easier to read $N \times N$ matrix form:

$$x = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix} \tag{3}$$

The reader can easily verify that this solution is indeed correct.

Problem 1 continued on next page...

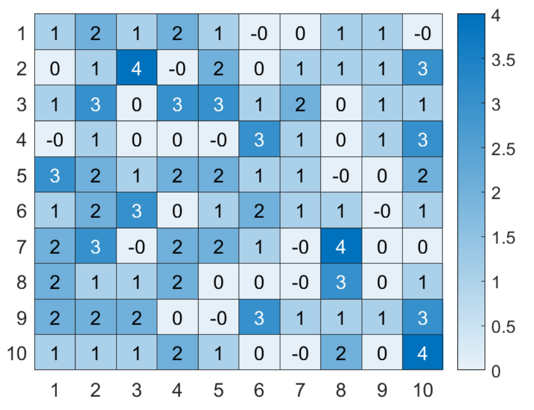
2

The steepest descent method and the conjugate gradient method were implemented in the functions called "solveSD" and "solveCG" to minimize quadratic forms $f(x) = \frac{1}{2}x^TAx - b^Tx + c$ with symmetric positive-definite A matrices. Since the solution to a system $Ax = b$ is also the minimum of a the quadratic form $f(x) = \frac{1}{2}x^TAx - b^Tx + c$, the steepest descent and the conjugate gradient methods allow to solve systems of linear equations. The functions solve SD and solve CG take a positive definite, symmetric matrix A and a column vector b as argument and output the solution of the system of equations. The steepest descent method is iterative whereas the conjugate gradient is a direct method. Both methods where tested on an example given during the lecture and give the correct result with a precision better than 10^{-10}

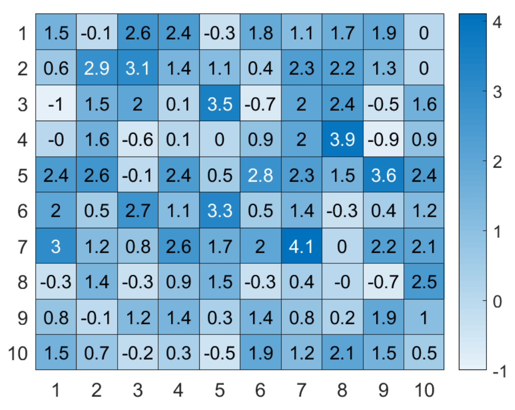
3

Both methods where used to solve two given 10×10 puzzles. To solve the puzzles, rather than using the function on the system $Ax = b$, they where applied to the system $AAx = Ab$ which is an equivalent system of equation if A is invertible which is the case here. The justification for doing this is that the matrix A isn't positive definite but the matrix AA is. For the steepest descent method, the solution indeed diverges if we don't multiply the system of equations by A on the left. For the conjugate gradient the correct solution is also found without multiplying the system of equations by A.

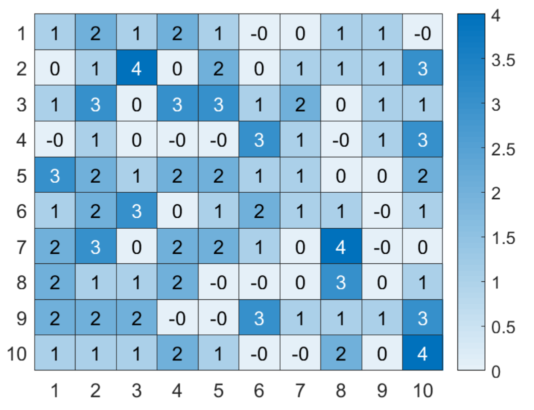
The solutions found for the two puzzles are given in a graphical way on figures 2



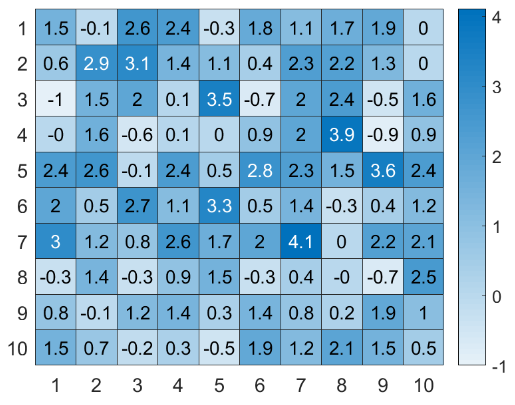
(a) First puzzle, steepest descent algorithm



(b) Second puzzle, steepest descent algorithm



(c) First puzzle, conjugate gradient algorithm



(d) Second puzzle provided, conjugate gradient algorithm

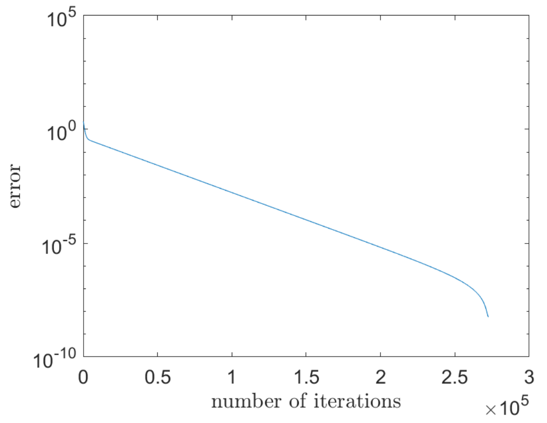
Figure 2: Graphical representation of the solutions for puzzles provided using steepest descent and conjugate gradient algorithms

4

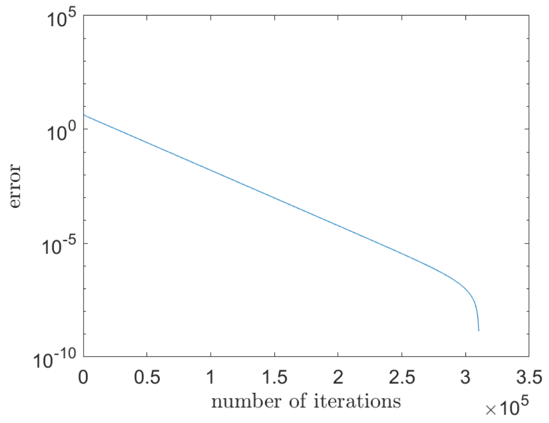
The convergence of speed of the two algorithms was analysed and compared. On figure 3 one can see the error of the solution as a function of the number of iteration. The error after n iterations was calculated by taking the norm of the difference of the result after n and the converged result. With the conjugate gradient method it is sufficient to perform a 100 iterations to achieve convergence whereas in the case of the steepest descent method the number of iterations needed to converge for these two puzzles is of the order 10^5. This is due to the fact that the conjugate gradient method is an exact method. The solution to these problems is of dimension 100. This means that after 100 iterations the found solution is exact up to a numeric error. In the case of the steepest descent algorithm, a lower bound for the convergence speed is given by:

$$\frac{f(x_i) - f(x)}{f(x_0) - f(x)} \leq \left(\frac{\kappa - 1}{\kappa + 1}\right)^i \tag{4}$$

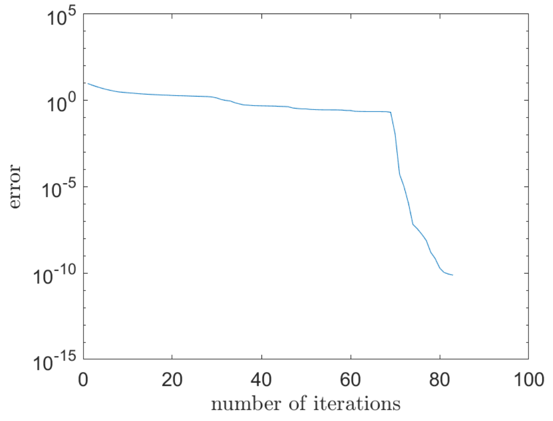
where $f(x)$ is the quadratic form given by a matrix A that needs to be positive definite and symmetric which is minimized, x_i is the approximated solution after i iterations and $\kappa = \frac{\lambda_max}{\lambda_min}$ is the ratio of the greatest and smallest eigenvalues of A. In the present case $\kappa = 0.9999462034$ which means that after $3 \cdot 10^5$ iterations the ratio $\frac{f(x_i)-f(x)}{f(x_0)-f(x)}$ must be smaller than 10^{-7} which is verified. Since the steepest descent method is iterative and we only have equation 4 as a convergence property, we can't conclude anything on the number of iterations required to reach a given precision. Indeed since equation 4 depends on $f(x_0) = -f(x)$ and x_0 is chosen randomly in the steepest descent function and can therefore be arbitrarily large, the only way to know how many iterations are needed is to do apply the function and observe the residuals after some number of iteration.



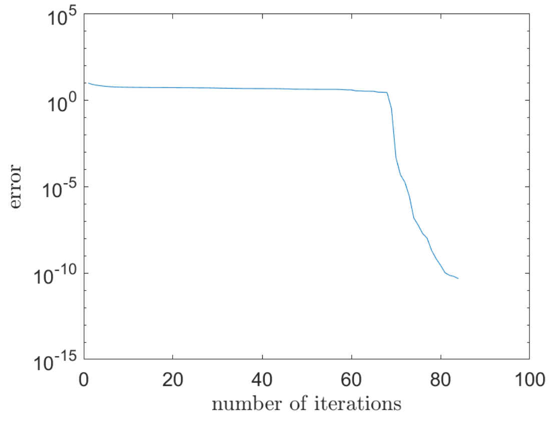
(a) First puzzle, steepest descent



(b) Second puzzle, steepest descent



(c) First puzzle, conjugate gradient



(d) Second puzzle, conjugate gradient

Figure 3: Plot of deviation from converged result as a function of the number of iteration for both puzzles and both algorithms

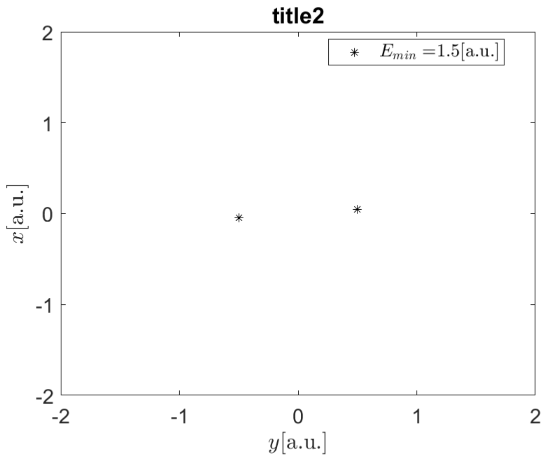
Problem 2

Exercise 2 This exercise deals with charged particles in a 2D harmonic potential. The nonlinear conjugate gradient method was implemented to find the minima of the potential energy of an ensemble of N charged particles. To find the minimum of the energy in one direction, the Newton-Raphson line search method was used.

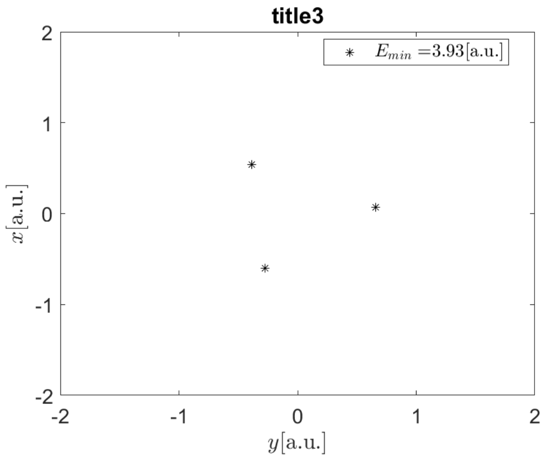
The implementation was tested on ensembles of $N = 2, 3, 4$ particles. The most stable configuration of the charged particles in the harmonic potential can be visualized on figure 4.

The energies of the most stable configurations for up to 10 particles are given in table 1.

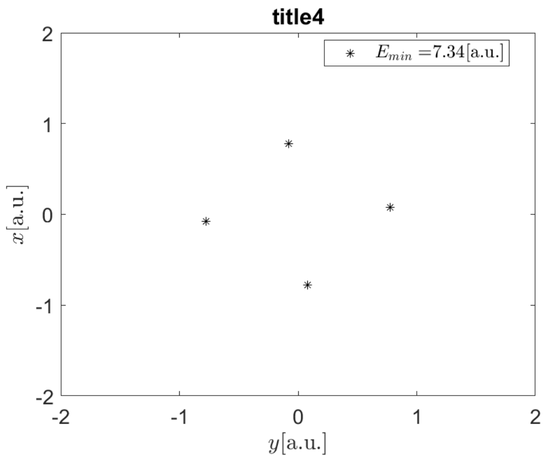
Problem 2 continued on next page...



(a) $N = 2$



(b) $N = 3$



(c) $N = 4$

Figure 4: Visualisation of the most stable configuration of charged particles in a harmonic 2D potential

N[ ]	5	6	7	8	9	10
E[a.u.]	11.69	16.83	22.67	29.35	36.85	44.88

Table 1: Value of energy of most stable configuration of electrons

Problem 3

Exercise 1 In this exercise the following over defined system of linear equations is considered:

$$\begin{pmatrix} 1 & 1 \\ 1 & 2 \\ 1 & 3 \end{pmatrix} \cdot \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} = \begin{pmatrix} 1 \\ 1.2 \\ 1.1 \end{pmatrix} \tag{5}$$

The least square solution of this equation was found using the Penrose matrix of matrix A. The found solution is:

$$\vec{x} = \begin{pmatrix} 1 \\ 0.05 \end{pmatrix}$$

It was then illustrated that this solution is indeed the one that minimizes the norm of the residual $r(x)$

Problem 3 continued on next page. . .

given by:

$$r(x) = ||Ax - b||$$

This illustration was done by plotting its magnitude in the vicinity of the solution $\vec{x}$. This was done using Matlab's function fcontour. The plot of the magnitude of $r(x)$ in the vicinity of $\vec{x}$ can be seen on figure 5.

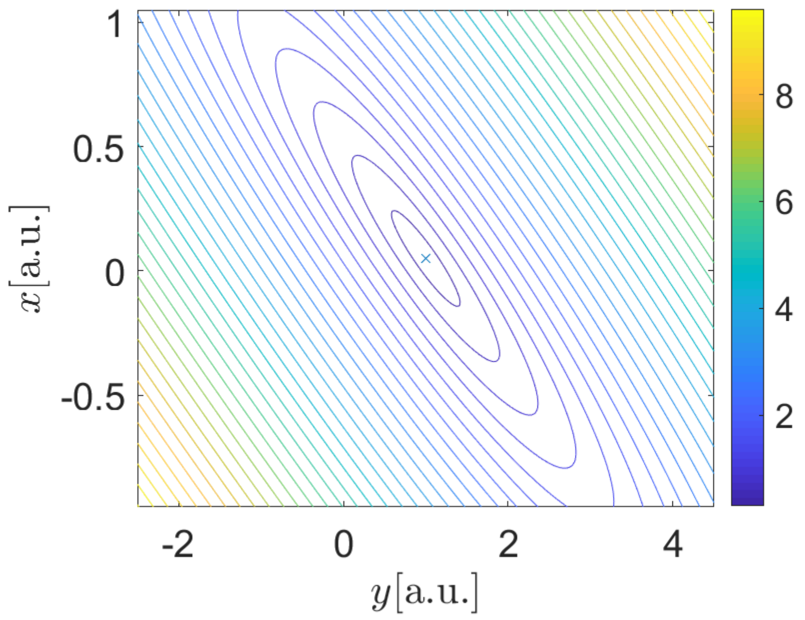


Figure 5: Magnitude of residual in the vicinity of the least square solution to system of equations 5

Problem 4

Exercise 2 In this exercise, singular value decomposition will be used for image compression. The singular value decomposition of a 312 by 220 image was performed and the singular values were plotted on a logarithmic scale. This can be seen on figure 6.

Singular value decomposition expresses an arbitrary matrix A in the form:

$$A = USV^\dagger \tag{6}$$

where U and V are unitary matrices and S is a diagonal matrix.

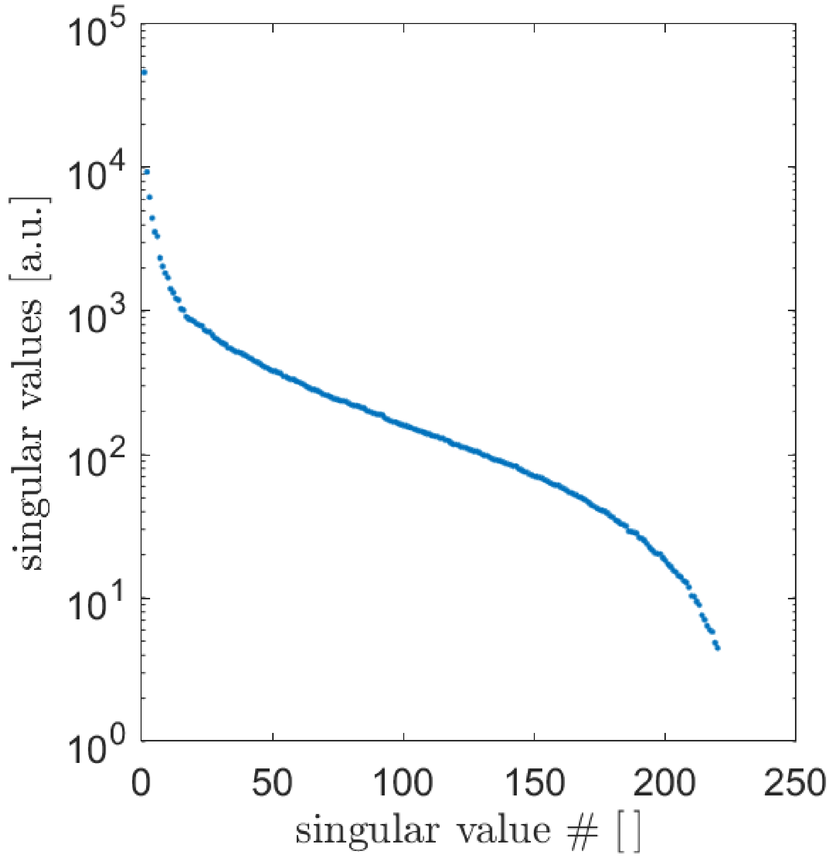


Figure 6: Plot of singular values of image as a function of their number (singular values are numbered in decreasing order)

The image was then approximated (compressed) by retaining only a certain number of the greatest singular values. This means that the smallest diagonal elements of matrix S were replaced by zeros in equation 6. The matrix A thus obtained isn't the original image anymore but a compressed version of it. An example of such compression on the image considered in this problem can be seen on figure 7 where all, 100, 20 and 4 singular values were retained. The first image retaining all singular values which is 220 is of course the same as the original image. 220 is also the rank of the matrix defined by the image. It is therefor a matrix of maximal rank because in a rectangular matrix there can only be as many independent lines as the minimum between the number of rows and columns which in this case is precisely 220.

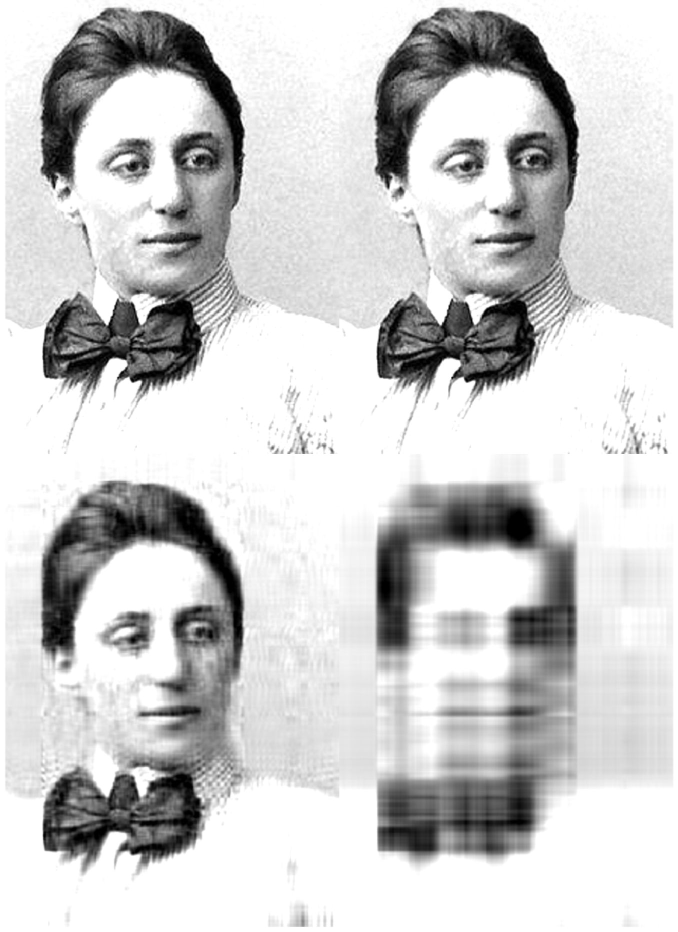


Figure 7: Original image on the upper left corner, compressed images retaining 100 singular values on the upper right corner, retaining 20 singular values on the lower left corner and retaining 4 singular values on the lower right corner

The quality of the images decreases with the number of singular values retained. It can be noticed that the quality of the image doesn't change much between the original image and the one with 100 singular values, it changes more between the image with 100 and the image with 20 singular values and it changes a lot more between the image with 20 value and the one with 4 values. This shows that it isn't so much the number of singular values that matters but rather that those that are neglected and set to zero would be small. This is confirmed by looking at figure 6. The value from #100 to #220 are relatively small compared to those between #1 and #20 but the values between #4 and #20 are relatively close to those between #1 and #4 which explains why the most compressed image on figure 7 is qualitatively different from the three others.

To store the compressed images, it takes as much memory as to store the N singular values of matrix S that are chose to be kept as well as the relevant value of U and V. The relevant values of U and V are those that intervene in the matrix product $U\mathcal{S}\mathcal{V}^\dagger$ where $\mathcal{S}$ is the old matrix S with N singular values set to zero. The reader can verify that since U and V are unitary and since the last $312-220$ columns of U are irrelevant values to start with (they don't intervene in the product $USV^\dagger$), for the case where N singular values are kept, there will be $312N - \frac{1}{2}(N^2 - N)$ relevant values in matrix U and $220N - \frac{1}{2}(N^2 - N)$ relevant values in matrix V. Adding to this the N singular value yield a total of $534N - N^2$ value that need to be stored. The values are doubles which means it will take up 64bits per value. As an example, it can be predicted that the

Problem 4 continued on next page. . .

compressed image for which 100 singular values are kept will take up $2'777'600$bits. For reference, the size of the original image once it is imported in Matlab is $4'392'960$bits . The compressed image is $63,2\%$ of the size of the original image and is visually hard to distinguish from the original image.

Problem 5

Exercise 3 In this exercise, the mean field approximation of the Hubbard model is considered. The calculations of the model are not performed in this exercise but the results are analysed. The considered Hamiltonian is:

$$\hat{\mathcal{H}} = -t \sum_{<i,j>,\sigma} a^{\dagger}_{i,\sigma} a_{j,\sigma} + U \sum_{i,\sigma} n_{i,\sigma} \langle n_{-\sigma} \rangle \tag{7}$$

The value of interest is the expectation value of the double occupancy which is plotted as a function of U/t on figure 8.

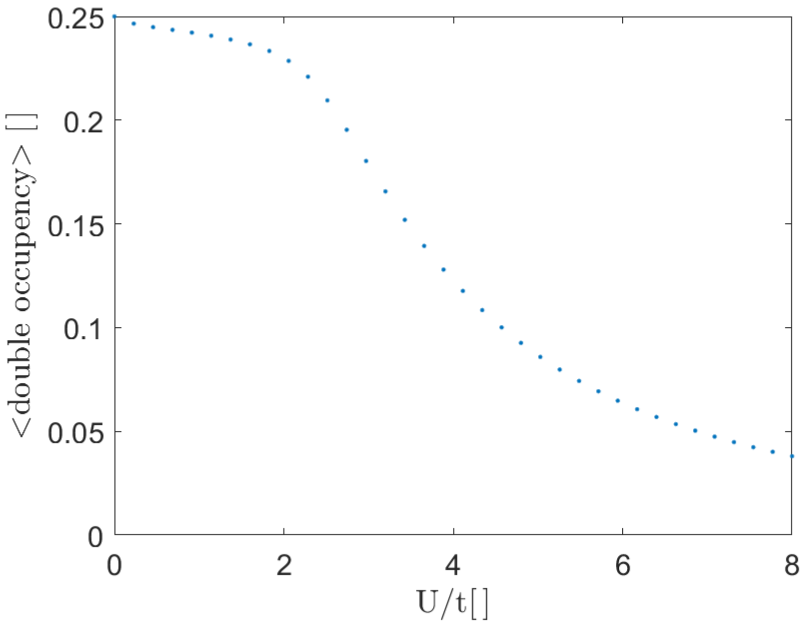


Figure 8: Expectation value of the double occupancy as a function of U/t

The trend of figure 8 fits with equation 7. Indeed, U and the sum that multiplies it both being positive, when U is large, having electrons of opposite spin from the mean costs energy proportional to U. If something costs energy it will or course be less likely. The sum multiplying U grows with each each spin going in the opposite direction from the mean. Typically it is expected that if U goes to infinity all the spins will point in the same direction and since Pauli's exclusion principle excludes the possibility that two spins being on the same site point in the same direction, there will be no double occupancy at all. On the other hand if U is equal to zero, the dynamic of spins on the lattice will be completely dictated by the first term is the Hamiltonian at equation 7. This means that the electron will jump winning energy $-t$ in is such a way as to minimize the energy of the system. I this case there will be non vanishing probability of having double occupancy on any given site. The more interesting case is what happens in between these two cases. It can be seen from the graph on figure 8 that the double occupancy expectation value depends on the ratio U/t. A way to understand this trend is that if there is a spin on each site all pointing in the same direction then the second term in the Hamiltonian is minimized but an opportunity lost to lower the energy of the system thanks to the first term. Indeed the first term can lower the energy of the system proportionally to t if electrons are allowed to jump but if there is a spin on each lattice site all pointing in the same direction then electrons can not jump because of Pauli's exclusion principle. An electron of spin up cannot jump to

Problem 5 continued on next page...

the neighbouring site if it is already occupied by a electron of spin up. A way to solve this is to have some double occupied sites which implies there will be some empty sites which in turn implies that there will be possibilities for at least some electrons (those next to the empty sites) to jump and lower the energy of the system. It quiet instinctive to see from here that if t is large relatively to U it will be worth having more empty states to allow for more jumps which means there will also be more double occupied states and vice versa for the case where U is large relative to t.