M4N9 PROJECT 2

Isa Majothi
CID: 00950286
The contexts of this report and associated codes are my own work unless otherwise stated

Autumn 2017

QUESTION 1

For a system of N sufficiently small particles moving slowly through a fluid, all of which are equally spaced along a horizontal line and moving directly upwards relative to this line, the following linear system is formed

$$\begin{pmatrix} M_{11} & M_{12} & \dots & M_{1N} \\ M_{21} & M_{22} & \dots & M_{2N} \\ \vdots & \vdots & \ddots & \vdots \\ M_{N1} & M_{N2} & \dots & M_{NN} \end{pmatrix} \begin{pmatrix} F_1 \\ F_2 \\ \vdots \\ F_N \end{pmatrix} = \begin{pmatrix} V_1 \\ V_2 \\ \vdots \\ V_N \end{pmatrix} \tag{1}$$

where each of the M_{ij} are 2×2 matrices, and F_i and V_i are 2×1 vectors representing the force on and velocity of particle i. In this project I will seek the solution F to the linear system $MF = V$ where $V_i = [0, 1]^T$ for $i = 1, \cdots, N$ and M is obtained using the provided MATLAB functions *Msetup* and *RPY2D*. Initially I will do this by using LU factorisation, but examining certain properties of M I will then aim to exploit these so to reduce the computational cost of solving the system.

Part (a)

After solving the system $MF = V$ as described above for different values of b and N (where b represents the spacing between neighbouring particles and N the total number of particles), I have included the values of $||F||_2$ for each case in Table 1, as well as plots of the y-components on the forces on each particle against x_i/x_N in Figures 1-6. Note that all values are given in standard form to 4 decimal places (d.p), and that the values in Table 1 can be found in the matrix *normsF1a* in my MATLAB code.

	$b = 2$	$b = 4$	$b = 10$
$N = 100$	4.5055×10^1	7.3557×10^1	1.1603×10^2
$N = 200$	5.6482×10^1	9.4408×10^1	1.5424×10^2
$N = 400$	7.1752×10^1	1.2219×10^2	2.0576×10^2

Table 1: Value of $||F||_2$ for different values of b and N

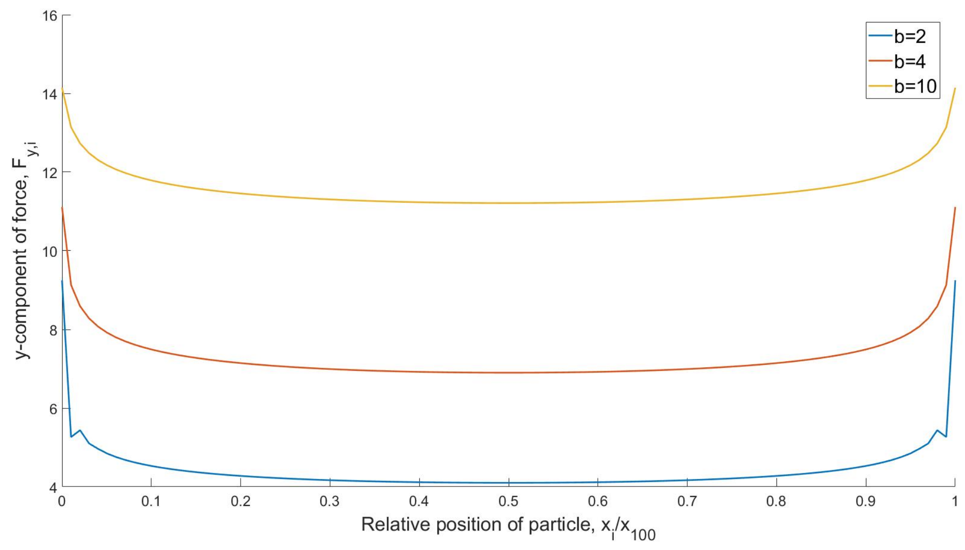

Figure 1: Plot of y-component of force against x_i/x_N with $N = 100$ and $b = 2, 4$ and 10

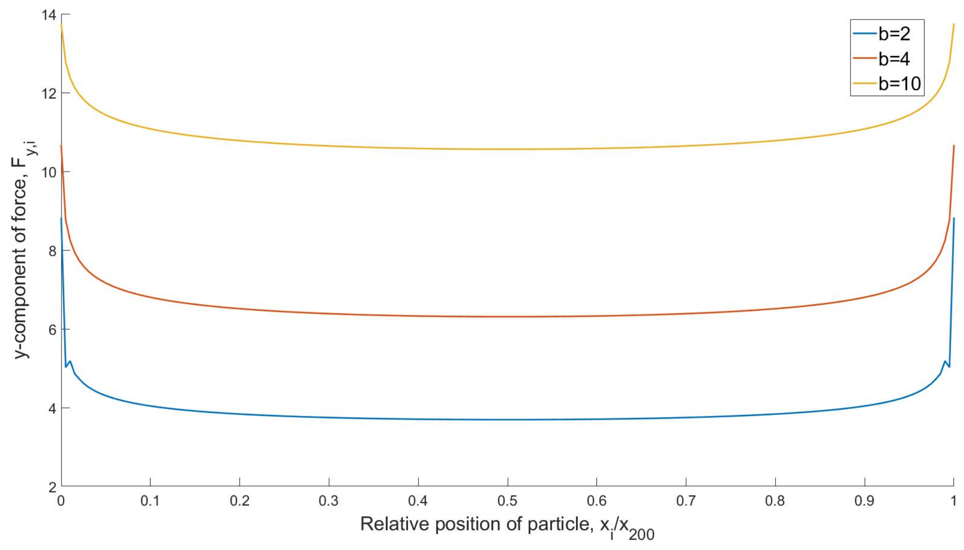

Figure 2: Plot of y-component of force against x_i/x_N with $N = 200$ and $b = 2, 4$ and 10

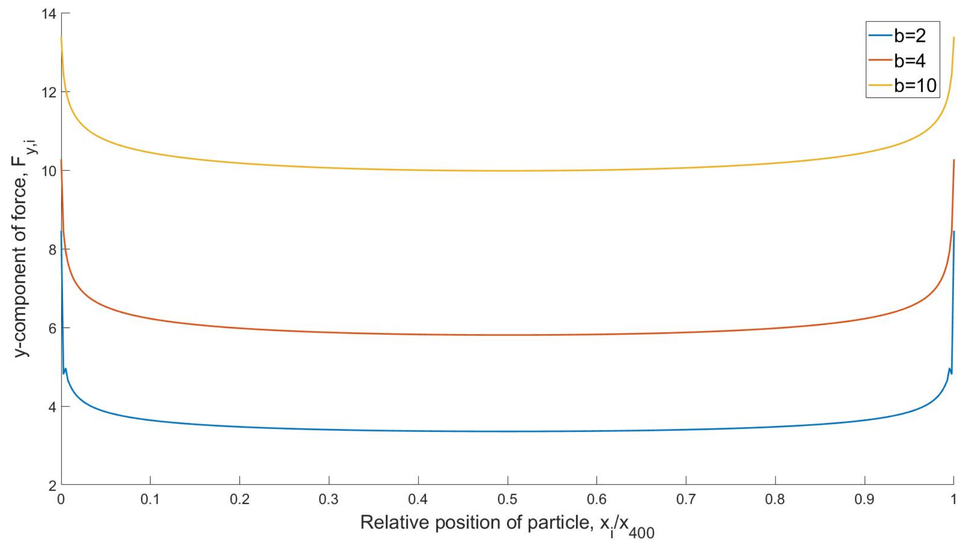

Figure 3: Plot of y-component of force against x_i/x_N with $N = 400$ and $b = 2, 4$ and 10

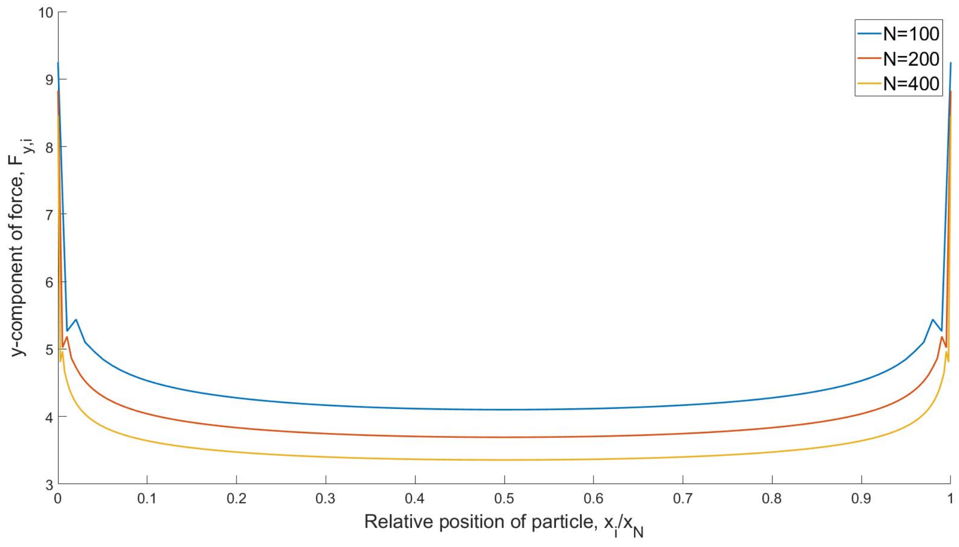

Figure 4: Plot of y-component of force against x_i/x_N with $b = 2$ and $N = 100, 200$ and 400

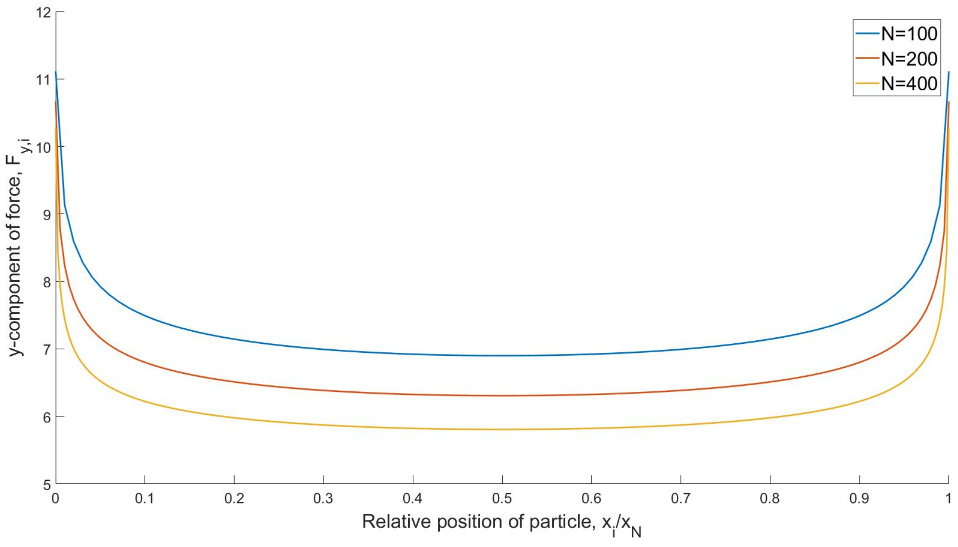

Figure 5: Plot of y-component of force against x_i/x_N with $b = 4$ and $N = 100, 200$ and 400

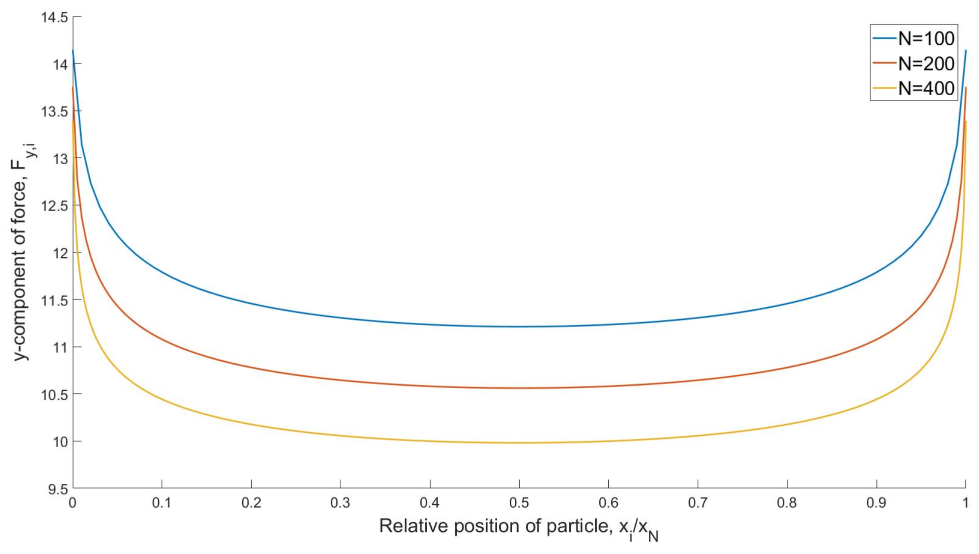

Figure 6: Plot of y-component of force against x_i/x_N with $b = 10$ and $N = 100, 200$ and 400

After analysing Table 1 and Figures 1-6, it is appears that increasing b increases the vertical forces acting on each particle, whilst increasing N decreases these forces. First looking at Table 1, we see that as we move along every row (increasing b for fixed N) as b increases from 2 to 10 $||F||_2$ increases for all three fixed values of N. Further evidence of this is reflected in Figures 1-3; in all three figures, the y-components of the force all increase across the entire domain as b increases.

On the contrary, increasing N leads to a decrease in the vertical components of the force acting on each particle. When fixing b and varying N between 100, 200 and 400, the force decreased for all three values of b, namely 2, 4 and 10. In Figures 4-6, all three plots demonstrate a reduction in the vertical forces acting on the particles as N increases. Furthermore, this decrease is most extreme for larger values of b, because Figure 4 shows that when $b = 2$ there is a much smaller differences between the forces when N =100, 200 and 400 compared with Figure 6 in which $b = 10$.

Note that considering $||F||_2$ is not a valid way to determine the effect N has on the force, even though Table 1 shows that for a fixed b the value of $||F||_2$ increases. These values are somewhat misleading because as N increases the length of the vector F increases (since $F \in \mathbb{R}^{2N}$), which would inevitably lead to an increase in its norm since there are more elements included in the calculation of the norm; even though $||F||_2$ may increase, Figures 4-6 clearly show that increasing N decreases the force acting on the particles, not increases these forces.

Part (b)

In order to assess the effect that increasing N has on the time taken for the system to be solved, I have recorded the times taken to find the solution of $MF = V$ using LU factorisation for different values of N. I recorded the times taken for when $b = 4$ and $N = 100, 200, 400, 800$ and 1600, which is stored in the vector *times1b* in my MATLAB code, and these times are reported in Table 2 in standard form to 4 d.p. To observe how this time behaves for large N, I used the MATLAB function *polyfit* on the log of these times against $\log(N)$ to determine the relationship between t and N as N becomes large.

N	Recorded Time
100	5.5457×10^{-2}
200	2.8193×10^{-1}
400	1.8881×10^{0}
800	2.0470×10^{1}
1600	2.2745×10^{2}

Table 2: Times taken to compute the LU decomposition of M and solve $MF = V$ for different values of N when $b = 4$

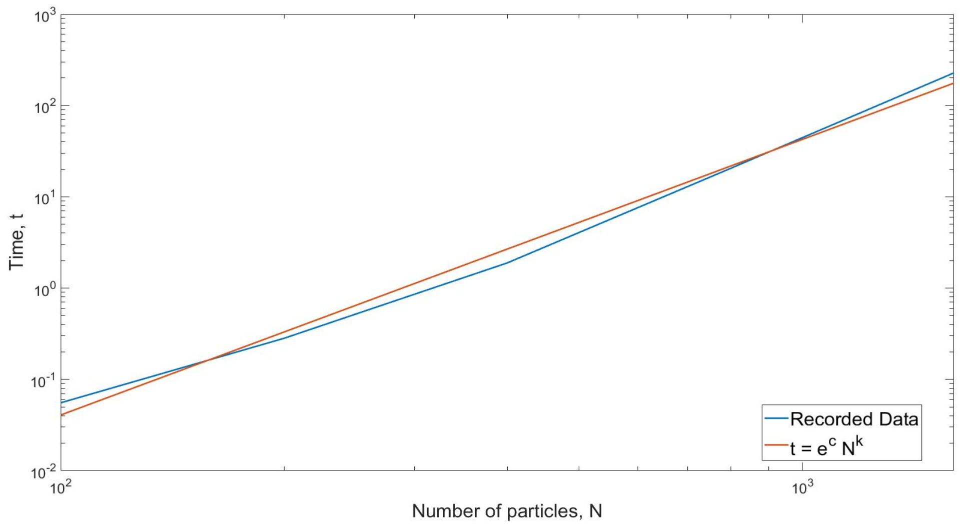

Figure 7: log-log plot of t against N using LU factorisation on M

To assess how the time t changes as N becomes large, I considered $\log(t)$ against $\log(N)$ so that if $t \sim N^k$ for large N then we would expect a linear relationship between $\log(t)$ and $\log(N)$, where the constant of proportionality is k. According to the log laws, this would mean

$$\log(t) = k \cdot \log(N) + c$$
$$\Rightarrow t \sim N^k \quad \text{for large } N$$

Consequently, to determine the behaviour of t for large N I plotted the data in Table 2 on a log-log scale against the function $t = e^c N^k$, where c and k can be found in the vector *coeff1b* in my MATLAB code.

The value of k obtained using *polyfit* was 3.0186×10^0 in standard form to to 4 d.p, and c was -1.7101×10^1 in standard form to 4 d.p. Drawing upon the ideas just discussed, this indicates that we do roughly observe $O(N^3)$ as N becomes large since it is the value of k that represents the way in which the operation cost changes with N and $k = 3$ when rounded to 1 d.p. Note that this is to be expected since the operation cost of applying LU factorisation is $O(N^3)$ as N becomes large.

Part (c)

For the cases where $b = 4$ and $N = 100, 200$ and 400, I have included the residual of each LU decomposition in Table 3 in standard form when rounded to 4 d.p, which can be found in the vector *residuals1c* in my MATLAB code. The provided MATLAB function *parpivgelim* computes the LU factorisation of M such that $LU = PM$ for some permutation matrix P, meaning that $P^T LU = M$ since $P^T P = I$ for any permutation matrix. As a result, the residual r of each LU decomposition is given by

$$r = ||P^T LU - M|| / (||L|| \cdot ||U||)$$

for any given system $MF = V$, where $|| \cdot ||$ is the matrix norm.

N	r
100	1.9748×10^{-16}
200	2.9340×10^{-16}
400	3.9603×10^{-16}

Table 3: Residual of the LU decomposition for different values of N when $b = 4$

Note that all three residuals are of $O(10^{-16})$, which is expected because MATLAB works to double precision and has machine epsilon $\epsilon = 10^{-16}$; by a theorem stated in lectures, we'd expect $r = O(\epsilon)$ so the values in Table 3 are reassuring.

Another statement from lectures tells us that using the LU decomposition to perform Gaussian elimination on a square matrix M is backward stable if $||L|| \cdot ||U|| = O(||M||)$. Again, for $b = 4$ and $N = 100, 200$ and 400 I have included $||L|| \cdot ||U||$ and $||M||$ for each case in Table 4, which are stored in the vectors *normsLU1c* and *normsM1c* respectively, where each value is given in standard form to 4 d.p.

N	$\|\|L\|\| \cdot \|\|U\|\|$	$\|\|M\|\|$
100	2.2050×10^{-1}	2.2045×10^{-1}
200	2.4803×10^{-1}	2.4800×10^{-1}
400	2.7558×10^{-1}	2.7557×10^{-1}

Table 4: Values of $||L|| \cdot ||U||$ and $||M||$ for different values of N when $b = 4$

The values in Table 4 provide numerical evidence of backward stability because in all three cases we see that $||L|| \cdot ||U|| = O(||M||)$ holds true, because when rounded to 3 d.p the values of $||L|| \cdot ||U||$ and $||M||$ are the same for $N = 100, 200$ and 400 in Table 4.

Question 2

Part (a)

Recall that the $2N \times 2N$ matrix M is of the form

$$\begin{pmatrix} M_{11} & M_{12} & \dots & M_{1N} \\ M_{21} & M_{22} & \dots & M_{2N} \\ \vdots & \vdots & \ddots & \vdots \\ M_{N1} & M_{N2} & \dots & M_{NN} \end{pmatrix}$$

Now the given approximation for each of the M_{ij} we are using is known as the *Rotne-Prager-Yamakawa* (RPY) *tensor*, which is defined by

$$M_{ij} = \begin{cases} \frac{1}{8\pi\eta r_{ij}}\left[\left(1+\frac{2a^2}{3r_{ij}^2}\right)I + \left(1-\frac{2a^2}{r_{ij}^2}\right)\frac{(X_i-X_j)(X_i-X_j)^T}{r_{ij}^2}\right] & \text{for} \quad r_{ij} > 2a \\ \frac{1}{6\pi\eta a}\left[\left(1-\frac{9r_{ij}}{32a}\right)I + \frac{3r_{ij}}{32a}\frac{(X_i-X_j)(X_i-X_j)^T}{r_{ij}^2}\right] & \text{for} \quad r_{ij} \leq 2a \end{cases} \tag{2}$$

where each term and variable is defined in the task.

Since we are considering the case where $X_i = [x_i, 0]^T$ for $i = 1, \cdots, N$, we observe that

$$(X_i - X_j)(X_i - X_j)^T = \begin{pmatrix} |x_i - x_j|^2 & 0 \\ 0 & 0 \end{pmatrix}$$

Taking this into account, and using the fact that $M_{ij} = \alpha I + \beta(X_i - X_j)(X_i - X_j)^T$ for all $i, j = 1, \cdots, N$ (where clearly $\alpha, \beta \in \mathbb{R}^+$) in the RPY tensor, we see that

$$M_{ij} = \begin{pmatrix} * & 0 \\ 0 & * \end{pmatrix} \tag{3}$$

where the entries along the diagonal represent positive real numbers.

It is evident that M_{ij} is defined like this regardless of the values of b or N, so by concatenating the M_{ij}'s accordingly we can conclude that M has a leading diagonal of non-zero entries followed by an alternating pattern of zero and non-zero diagonals both above and below the main diagonal, as is demonstrated in (4)

$$M = \begin{pmatrix} * & 0 & * & \cdots & \cdots & \cdots & * & 0 \\ 0 & * & 0 & \ddots & & & & * \\ * & 0 & * & \ddots & \ddots & & & \vdots \\ \vdots & \ddots & \ddots & \ddots & \ddots & \ddots & & \vdots \\ \vdots & & \ddots & \ddots & \ddots & \ddots & \ddots & \vdots \\ \vdots & & & \ddots & \ddots & * & 0 & * \\ * & & & & \ddots & 0 & * & 0 \\ 0 & * & \cdots & \cdots & \cdots & * & 0 & * \end{pmatrix} \tag{4}$$

Part (b)

After having observed the lattice-like structure of the matrix M in (4), we can exploit this to reduce the size of our problem and hence the operation cost (since we observed that, when using LU factorisation, the time taken to solve the system increases like N^3 for large N in Question 1b). Note that $M \in \mathbb{R}^{2N\times 2N}$, meaning that M has $4N^2$ entries, half of which are zeros and the other half non-zeros by (3). Furthermore, after the findings made about M in Part (a), the system $MF = V$ can be viewed as

$$\begin{pmatrix} * & 0 & * & \cdots & \cdots & 0 & * & 0 \\ 0 & * & 0 & \ddots & & & & * \\ * & 0 & * & \ddots & \ddots & & & 0 \\ \vdots & \ddots & \ddots & \ddots & \ddots & \ddots & & \vdots \\ \vdots & & \ddots & \ddots & \ddots & \ddots & \ddots & \vdots \\ 0 & & & \ddots & \ddots & * & 0 & * \\ * & & & & \ddots & 0 & * & 0 \\ 0 & * & 0 & \cdots & \cdots & * & 0 & * \end{pmatrix} \begin{pmatrix} F_{11} \\ F_{12} \\ F_{21} \\ F_{22} \\ \vdots \\ \vdots \\ F_{N1} \\ F_{N2} \end{pmatrix} = \begin{pmatrix} V_{11} \\ V_{12} \\ V_{21} \\ V_{22} \\ \vdots \\ \vdots \\ V_{N1} \\ V_{N2} \end{pmatrix} \tag{5}$$

where F_{i1} and V_{i1} correspond to the x-component of the force and velocity of particle i respectively, and likewise F_{i2} and V_{i2} are the respective y-components.

Upon closer examination, we observe that in the linear system of equations, the x-components of F and V only correspond to the elements of M in the rows and columns with odd indexes (see red entries in (5)), and similarly for the y-components with the even indexes (see blue entries in (5)). So rather than considering (5) as a $2N \times 2N$

linear system, it can be thought of as two $N \times N$ systems, one for the x-components and the other for the y-components since the equations are completely independent of one another due to the lattice-like structure of M. Note that this makes sense since we are solving in orthogonal directions, hence we would expect there to be independent systems in the x- and y-directions.

In order to find the optimal permutation matrix $P \in \mathbb{R}^{2N\times 2N}$ that will allow us to permute the matrix M to give these two $N \times N$ systems, I will find P for $N = 2$ and $N = 3$, before extending this to general N. Note that I will be seeking P such that

$$PMP^T = \begin{pmatrix} M_1 & 0 \\ 0 & M_2 \end{pmatrix} \tag{6}$$

where M_1, M_2 and 0 are all $N \times N$ matrices, and more specifically that M_1 consists of the red entries of M in (5) so that they occur *in this order*, and similarly for M_2 and the blue entries of M.

Using (6) and the fact that $P^TP = I$, this changes our linear system as follows

$$\begin{aligned} MF &= V \\ PMF &= PV \\ PM(P^TP)F &= PV \\ (PMP^T)(PF) &= PV \end{aligned} \tag{7}$$

Finding P for $N = 2$

For any value of b, when $N = 2$ the system $MF = V$ will be of the form

$$MF = \begin{pmatrix} a_1 & 0 & a_2 & 0 \\ 0 & a_1 & 0 & a_3 \\ a_2 & 0 & a_1 & 0 \\ 0 & a_3 & 0 & a_1 \end{pmatrix} \begin{pmatrix} F_{11} \\ F_{12} \\ F_{21} \\ F_{22} \end{pmatrix} = \begin{pmatrix} V_{11} \\ V_{12} \\ V_{21} \\ V_{22} \end{pmatrix} = V$$

due to the recurrence of the M_{ij}'s (i.e. $M_{ij} = M_{ji}$ for $i, j = 1, \cdots, N$, which leads to M being symmetric), where a_i are positive real numbers for $i = 1, \dots, 3$.

Since we are aiming to permute M so it is of the form in (6), whereby the relevant x-component coefficients are in M_1 and the y-component coefficients are in M_2, I have defined P to be

$$P = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

We can now compute the components of (7) as follows

$$\begin{aligned} PMP^T &= \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} a_1 & 0 & a_2 & 0 \\ 0 & a_1 & 0 & a_3 \\ a_2 & 0 & a_1 & 0 \\ 0 & a_3 & 0 & a_1 \end{pmatrix} \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \\ &= \dots \\ &= \begin{pmatrix} a_1 & a_2 & 0 & 0 \\ a_2 & a_1 & 0 & 0 \\ 0 & 0 & a_1 & a_3 \\ 0 & 0 & a_3 & a_1 \end{pmatrix}, \text{which is of the form in (6)} \end{aligned}$$

$$PF = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} F_{11} \\ F_{12} \\ F_{21} \\ F_{22} \end{pmatrix} = \begin{pmatrix} F_{11} \\ F_{21} \\ F_{12} \\ F_{22} \end{pmatrix}$$

$$PV = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} V_{11} \\ V_{12} \\ V_{21} \\ V_{22} \end{pmatrix} = \begin{pmatrix} V_{11} \\ V_{21} \\ V_{12} \\ V_{22} \end{pmatrix}$$

Now expressing this system in the form of (7), our linear system becomes

$$\begin{pmatrix} a_1 & a_2 & 0 & 0 \\ a_2 & a_1 & 0 & 0 \\ 0 & 0 & a_1 & a_3 \\ 0 & 0 & a_3 & a_1 \end{pmatrix} \begin{pmatrix} F_{11} \\ F_{21} \\ F_{12} \\ F_{22} \end{pmatrix} = \begin{pmatrix} V_{11} \\ V_{21} \\ V_{12} \\ V_{22} \end{pmatrix}$$

clearly showing that the x-components and y-components have effectively been separated into two independent 2×2 linear systems.

Finding P for $N = 3$

For any value of b, when $N = 3$ the system $MF = V$ will be of the form

$$MF = \begin{pmatrix} a_1 & 0 & a_2 & 0 & a_3 & 0 \\ 0 & a_1 & 0 & a_4 & 0 & a_5 \\ a_2 & 0 & a_1 & 0 & a_2 & 0 \\ 0 & a_4 & 0 & a_1 & 0 & a_4 \\ a_3 & 0 & a_2 & 0 & a_1 & 0 \\ 0 & a_5 & 0 & a_4 & 0 & a_1 \end{pmatrix} \begin{pmatrix} F_{11} \\ F_{12} \\ F_{21} \\ F_{22} \\ F_{31} \\ F_{32} \end{pmatrix} = \begin{pmatrix} V_{11} \\ V_{12} \\ V_{21} \\ V_{22} \\ V_{31} \\ V_{32} \end{pmatrix} = V$$

again due to the recurrence of the M_{ij}'s, where a_i are non-zero real numbers for $i = 1, \ldots, 5$.

In order to obtain a linear system resembling that in (6), I have defined P to be

$$P = \begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{pmatrix}$$

We can now compute the components of (7) as follows

$$\begin{aligned} PMP^T &= \begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} a_1 & 0 & a_2 & 0 & a_3 & 0 \\ 0 & a_1 & 0 & a_4 & 0 & a_5 \\ a_2 & 0 & a_1 & 0 & a_2 & 0 \\ 0 & a_4 & 0 & a_1 & 0 & a_4 \\ a_3 & 0 & a_2 & 0 & a_1 & 0 \\ 0 & a_5 & 0 & a_4 & 0 & a_1 \end{pmatrix} \begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{pmatrix} \\ &= \ldots \\ &= \begin{pmatrix} a_1 & a_2 & a_3 & 0 & 0 & 0 \\ a_2 & a_3 & a_2 & 0 & 0 & 0 \\ a_3 & a_2 & a_1 & 0 & 0 & 0 \\ 0 & 0 & 0 & a_1 & a_4 & a_5 \\ 0 & 0 & 0 & a_4 & a_1 & a_4 \\ 0 & 0 & 0 & a_5 & a_4 & a_1 \end{pmatrix}, \text{which is of the form in (6)} \end{aligned}$$

$$PF = \begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} F_{11} \\ F_{12} \\ F_{21} \\ F_{22} \\ F_{31} \\ F_{32} \end{pmatrix} = \begin{pmatrix} F_{11} \\ F_{21} \\ F_{31} \\ F_{12} \\ F_{22} \\ F_{32} \end{pmatrix}$$

$$PV = \begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} V_{11} \\ V_{12} \\ V_{21} \\ V_{22} \\ V_{31} \\ V_{32} \end{pmatrix} = \begin{pmatrix} V_{11} \\ V_{21} \\ V_{31} \\ V_{12} \\ V_{22} \\ V_{32} \end{pmatrix}$$

Now expressing this system in the form of (7), our linear system becomes

$$\begin{pmatrix} a_1 & a_2 & a_3 & 0 & 0 & 0 \\ a_2 & a_3 & a_2 & 0 & 0 & 0 \\ a_3 & a_2 & a_1 & 0 & 0 & 0 \\ 0 & 0 & 0 & a_1 & a_4 & a_5 \\ 0 & 0 & 0 & a_4 & a_1 & a_4 \\ 0 & 0 & 0 & a_5 & a_4 & a_1 \end{pmatrix} \begin{pmatrix} F_{11} \\ F_{21} \\ F_{31} \\ F_{12} \\ F_{22} \\ F_{32} \end{pmatrix} = \begin{pmatrix} V_{11} \\ V_{21} \\ V_{31} \\ V_{12} \\ V_{22} \\ V_{32} \end{pmatrix}$$

clearly showing that the x-components and y-components have effectively been separated into two independent 3×3 linear systems.

Finding P for general N

After having found the permutation matrix that modifies the system so that it is of the form in (7) in the cases when $N = 2$ and $N = 3$, and noticing the pattern in the way that P is constructed to give this permuted system, the permutation matrix for general N can be defined as follows

$$\begin{aligned} P &= (p_{kl}) \in \mathbb{R}^{2N \times 2N} \\ &= \begin{cases} (p_{k,2k-1}) = 1 & \text{for} \quad k = 1, \cdots, N \\ (p_{k+N,2k}) = 1 & \text{for} \quad k = 1, \cdots, N \\ 0 & \text{otherwise} \end{cases} \end{aligned} \tag{8}$$

After noting the symmetry observed in M when $N = 2$ and $N = 3$, applying this permutation matrix P to the system in (5) so that it is of the form in (7) yields

$$\begin{pmatrix} a_1 & a_2 & \cdots & \cdots & a_N & & & & & \\ a_2 & a_1 & \ddots & & \vdots & & & & & \\ \vdots & a_2 & \ddots & \ddots & \vdots & & & 0 & & \\ \vdots & & \ddots & \ddots & a_2 & & & & & \\ a_N & \cdots & \cdots & a_2 & a_1 & & & & & \\ & & & & & a_1 & a_{N+1} & \cdots & \cdots & a_{2N-1} \\ & & & & & a_{N+1} & a_1 & \ddots & & \vdots \\ & & 0 & & & \vdots & a_{N+1} & \ddots & \ddots & \vdots \\ & & & & & \vdots & & \ddots & \ddots & a_{N+1} \\ & & & & & a_{2N-1} & \cdots & \cdots & a_{N+1} & a_1 \end{pmatrix} \begin{pmatrix} F_{11} \\ F_{21} \\ \vdots \\ \vdots \\ F_{N1} \\ F_{12} \\ F_{22} \\ \vdots \\ \vdots \\ F_{N2} \end{pmatrix} = \begin{pmatrix} V_{11} \\ V_{12} \\ \vdots \\ \vdots \\ V_{N1} \\ V_{12} \\ V_{22} \\ \vdots \\ \vdots \\ V_{N2} \end{pmatrix} \tag{9}$$

which clearly illustrates how the x- and y-components of the system can be separated into two $N \times N$ linear systems.

As mentioned before, M_1 contains the coefficients corresponding to the x-components of the system, and M_2 the respective coefficients for the y-components. However, it is worth noting that the choice of P to give us a system where PMP^T is in the form of (6) is not unique; I defined P as in (8) deliberately in order to avoid ever having to perform cumbersome matrix multiplications involving $2N \times 2N$ matrices, particularly for large N.

When looking at PMP^T in the cases where $N = 2$ and $N = 3$, it is clear that the entries in the rows and columns with odd indexes have effectively been squashed into the top-left $N \times N$ entries to give M_1 whilst preserving their order of appearance, and similarly for M_2 in the bottom-right $N \times N$ section. As a result, extending this to general N tells us that the matrices M_1 and M_2 are readily obtainable simply by grouping together the entries in the odd rows without changing the order in which they appear, and similarly for the even rows.

As was also seen in the examples when $N = 2$ and $N = 3$, the vectors F and V are also modified when multiplied by P on the left. When multiplying by P, both vectors are permuted in the same way whereby the N x-components are stored in ascending order in the first N elements of the vectors, and below this the N y-components are stored in ascending order in the last N entries. Note that how to readily obtain M_1 and M_2 from M, as well as how to modify F and V so that the system is of the form in (7), can easily be visualised looking at (9).

Question 3

Following the observations made in Question 2, the system $MF = V$ can be solved much more rapidly by considering two $N \times N$ linear systems. Since we are considering the case where $V_i = [0, 1]^T$, this means that the x-components of the forces acting on each particle will all be equal to 0 regardless of the matrix M_1 i.e. $F_{i1} = 0$ for $i = 1, \ldots, N$. Consequently, only the $N \times N$ system involving M_2 needs to actually be computed, so in this question I have set $F_{i1} = 0$ for $i = 1, \ldots, N$ for this reason, and have only solved the system for M_2 in my code.

Note that in order to ensure that the solutions obtained in Question 3 are identical to those from Question 1b, I compared the norms of each solution F for the different values of N tested to ensure that they were equivalent within a certain tolerance that I specified to be 10^{-13}; for reference, this was done by comparing the vector *normsF1b* with *normsF3a* and *normsF3b*, as can be seen in the vectors *checker3a* and *checker3b*. I also observed the plots of the y-component of force against x_i/x_N for each method, and found that they were the same.

Part (a)

Firstly, I hoped to solve the system $MF = V$ more rapidly by still using LU factorisation, but by using it on $M_2 \in \mathbb{R}^{N\times N}$ as opposed to $M \in \mathbb{R}^{2N\times 2N}$. For the cases discussed in Question 1b, namely where $b = 4$ and $N = 100, 200, 400, 800$ and 1600, I recorded the times taken to solve the system and stored them in the vector *times3a* in my MATLAB code, and these are recorded in standard form to 4 d.p in Table 5. I also plotted these times against N on a log-log scale in Figure 8, again using *polyfit* to determine how the time t changes for N large as described in Question 1b.

N	Recorded Time	Speed-Up
100	1.2280×10^{-2}	4.7083×10^{0}
200	5.3249×10^{-2}	5.3207×10^{0}
400	2.7513×10^{-1}	6.8942×10^{0}
800	1.7476×10^{0}	1.1709×10^{1}
1600	2.0267×10^{1}	1.1325×10^{1}

Table 5: Times taken to compute the LU decomposition of M_2 and solve $MF = V$ for different values of N when $b = 4$, and the speed-up compared with Table 2

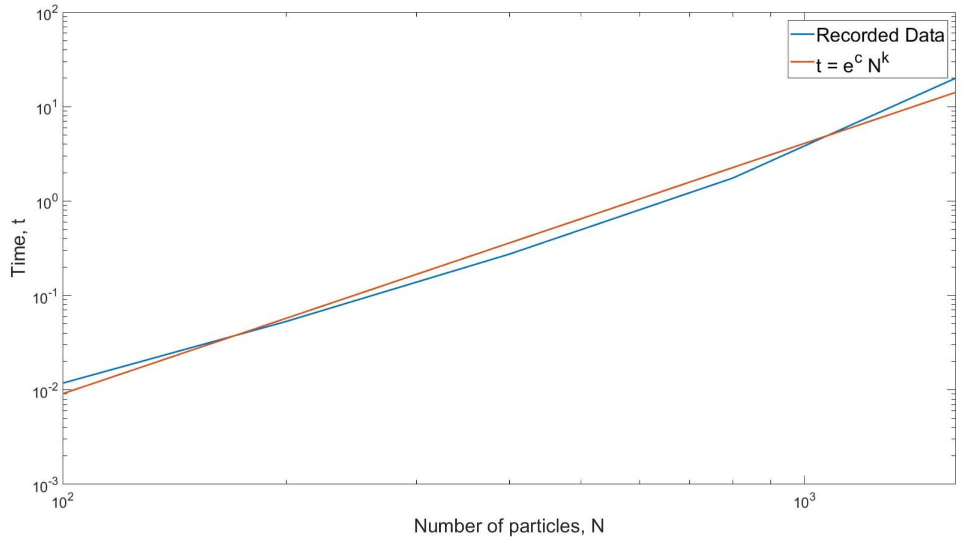

Figure 8: log-log plot of t against N using LU factorisation on M_2

It is clear from comparing the times recorded in Tables 2 and 5 that solving the system by applying the LU decomposition to M_2 rather than M produces a substantial increase in speed (and therefore reduction in operation cost). Even in the smallest value of $N = 100$, solving the system by applying LU factorisation to M_2 induces a speed-up of approximately 5, and this speed-up increases to approximately 11 for $N = 800$ and 1600.

Furthermore, upon using *polyfit* to determine the behaviour of t for large N, I found that solving only for M_2 gave $k = 2.6516 \times 10^0$ in standard form to 4 d.p (see *coeff3a* in my MATLAB code), which is significant for large N when compared to the value of $k = 3.0186 \times 10^0$ found in Question 1b; this is further evidence of the speed-up given by using this method to solve $MF = V$ over the method used in Question 1.

Part (b)

After examining M_2 as it appears in (9), it is clear that M_2 is a symmetric matrix whose entries decrease as we move along the first row i.e. $a_1 > a_{N+1} > \ldots > a_{2N-1}$ (this can be deduced by the definition of M_{ij} in (2)). Assuming that M_2 is symmetric positive definite i.e. that $x^T M_2 x > 0$ for all $x \in \mathbb{R}^N, x \neq 0$, we can alternatively use the *Levinson* algorithm to solve the system $MF = V$ by applying the algorithm as it is presented to us to the system $M_2F_2 = V_2$, rather than LU factorisation. Rather than using LU decomposition, which requires $O(N^3)$ flops for N large, the Levinson algorithm only requires $O(N^2)$ flops for large N, which should result in a large increase in efficiency.

Furthermore, I opted for this algorithm over the other algorithms presented to us in Chapter 4 of *Matrix Computations* by Golub and Van Loan because it can be used to solve any square linear system $Ax = b$ provided that A is a symmetric positive definite matrix, whereas other algorithms involved other conditions on b. Note that in implementing this algorithm, I used the code from *Matrix Computations* as it was published, and hereby do not claim the implemented Levinson algorithm coded in the MATLAB function *levinsonSystemSolver* to be my own.

I used Levinson's algorithm on M_2 to solve $MF = V$, again fixing $b = 4$ and letting $N = 100, 200, 400, 800$ and 1600. The times taken to obtain the solution using this method were stored in the vector *times3b* in my MATLAB code, and are reported in standard form to 4 d.p in Table 6. Again, I also plotted these times against N on a log-log scale in Figure 9, using *polyfit* to gain insight into how the time t changes as N becomes large. Note that I only used polyfit on the points $N = 400, 800$ and 1600 since the readings for the times when N was relatively small were too prone to change.

N	Recorded Time	Speed-Up
100	1.0961×10^{-3}	5.0596×10^{1}
200	3.1019×10^{-3}	9.0889×10^{1}
400	7.0671×10^{-3}	2.6716×10^{2}
800	1.7081×10^{-2}	1.1984×10^{3}
1600	5.0590×10^{-2}	4.4960×10^{3}

Table 6: Times taken to use the Levinson algorithm on M_2 and solve $MF = V$ for different values of N when $b = 4$, and the speed-up compared with Table 2

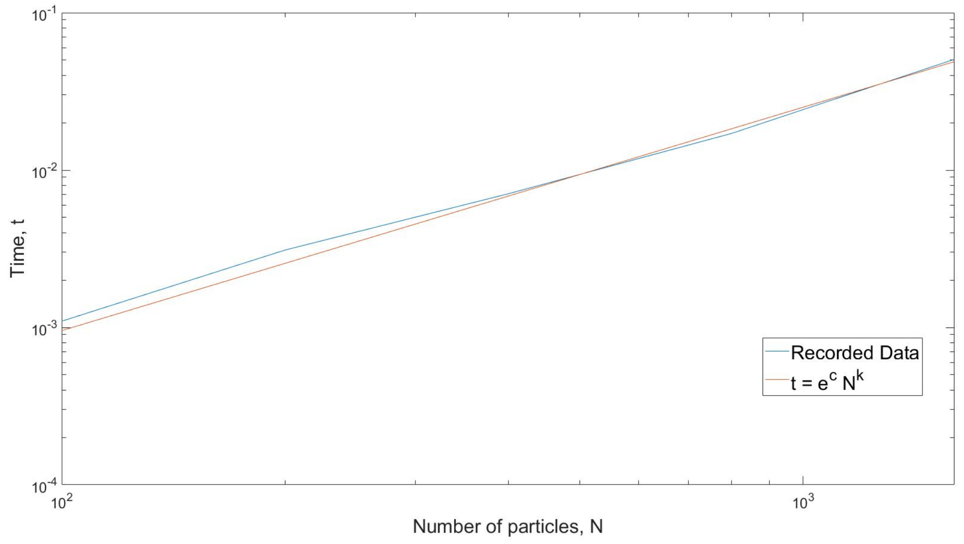

Figure 9: log-log plot of t against N using Levinson's algorithm on M_2

After comparing the times in Tables 2 and 6 for each value of N , using Levinson's algorithm produces a monumental speed-up when compared to the method used in Question 1b. Whilst Table 5 shows speed-ups of $O(10^1)$ when $N = 800$ and 1600, we already see a speed-up of this magnitude in the case when $N = 100$ in Table 6, and as N becomes large this speed-up increases dramatically such that it is of $O(10^3)$ when $N = 800$, let alone $N = 1600$. It is evident that whilst performing LU factorisation on M_2 resulted in a notable speed-up compared with the times observed in Question 1b, using Levinson's algorithm trumps both by some distance, reflected in the speed-ups in Table 6 and also when comparing these to the speed-ups in Table 5.

In addition to this, when using *polyfit* I obtained a value of $k = 1.4198 \times 10^0$ to 4 d.p in standard form (see *coeff3b* in my MATLAB code). When compared to the values of k found in Questions 1b and 3a, it is significantly smaller and further helps to demonstrate the benefit of using Levinson's algorithm to solve the system $MF = V$ over LU factorisation given the fact that we can permute M to give M_2 as we saw in Question 2 where M_2 is symmetric positive definite. Finally note that this computational saving is not surprising given that Levinson's algorithm requires $O(N^2)$ flops for large N, whereas LU decomposition requires $O(N^3)$.