Computational Linear Algebra Coursework 2

–

CID:01724711

Question 2

–

2a

$C = (xy^T)^k = (xy^T)\ldots(xy^T) = x(y^Tx)\ldots(y^Tx)y^T = (y^Tx)^{k-1}(xy^T)$ \ Algorithm \ · Calculate $\alpha = y^Tx \quad (N_{FLOPS} = 2n - 1)$ \ · Calculate $\beta = \alpha^{k-1} \quad (N_{FLOPS} = k - 2)$ \ · Calculate $A = xy^T \quad (N_{FLOPS} = n^2)$\ · Return $\beta A \quad (N_{FLOPS} = n^2)$\ Total $N_{FLOPS} = 2n - 1 + k - 2 + n^2 + n^2 = 2n^2 + 2n + k - 3 \sim \mathcal{O}(n^2)$

2b

Method 1)\ We can write $BA = (Ba_1 \ldots Ba_n)$ where $(Ba_i)_j$ requires $2m - 1$ flops. Since there are m values to count in (Ba_i) and n to count in BA we get a flop count of $nm(2m-1)$. Writing $C = BA$ where C has dimensions $m \times n$. We use the same reasoning but with different dimensions to obtain a flop count of $n^2(2m-1)$ for A^TC. Thus the total flop count for method 1 is $nm(2m-1) + n^2(2m-1)$.\ Method 2) \ We use the same reasoning as above but again changing the matrix dimensions accordingly to get $mn(2m-1)$ flop counts for A^TB and $n^2(2m-1)$ for $C'A$ when writing $C' = A^TB$. This gives a total of $mn(2m-1) + n^2(2m-1)$ flops. \ We can see that the number of flops for both method is the same hence for no values of m and n does the second algorithm ever have less flops.

2c

$A = (P + iQ)(R + iS) = PR - QS + i(QR + PS) \Rightarrow Re(A) = PR - QS, Im(A) = QR + PS$. The three matrix multiplications we choose are $T = (P+Q)(R-S)$, QR and PS. \ T: $P + Q$ and $R - S$ both require m^2 flops and then the matrix multiplication step requires $m^2(2m-1)$ again using the same approach as in 2b. Total $N_{FLOPS} = 2m^2 + m^2(2m-1)$\ QR: $m^2(2m-1)$ flops \ PS: $m^2(2m-1)$ flops \ $Re(A)$: A matrix subtraction and addition of dimensions $m \times m$ gives $2m^2$ flops \ $Im(A)$: A matrix addition of dimensions $m \times m$ gives m^2 flops \ TOTAL flops for $Re(A)$ and $Im(A) = 2m^2 + m^2(2m-1) + m^2(2m-1) + m^2(2m-1) + 2m^2 + m^2 = 2m^2(3m-1)$

Question 3

–

Let $A = \begin{bmatrix} a & b \\ c & d \end{bmatrix}$. We seek to solve $det(A - \lambda I) = 0$. This gives $\begin{vmatrix} a-\lambda & b \\ c & d-\lambda \end{vmatrix} = 0$

$\Rightarrow (a-\lambda)(b-\lambda) - bc = 0 \Rightarrow \lambda^2 - (a+d)\lambda + (ad - bc) = 0 \Rightarrow \lambda^2 - trace(A)\lambda + det(A) = 0 \Rightarrow \lambda = \frac{trace(A) \pm \sqrt{trace(A)^2 - 4det(A)}}{2}$

3a

We know that an algorithm $\tilde{f}$ for f is stable if for each $x \in X$, $\frac{||\tilde{f}(x) - f(\tilde{x})||}{||f(\tilde{x})||} = \mathcal{O}(\epsilon)$, there exists $\frac{||\tilde{x} - x||}{||\tilde{x}||} = \mathcal{O}(\epsilon)$ \ Specifically for this algorithm we have $f\colon \mathbb{C}^{2\times 2} \to \mathbb{C}^2$. For it to be forward stable we need to have that if for each matrix A with normed difference of the floating point implementation calculated vector of eigenvalues of true A and the true theoretical vector of eigenvalues of a perturbed A scaled by the norm of the true theoretical vector of eigenvalues of the perturbed A being accurate to machine precision there needs to exists a perturbed matrix of A such that the normed difference of $\tilde{A}$ and A scaled by the norm of A is also of order machine epsilon. So if we denote $[\lambda_1, \lambda_2]$ the floating point implementation calculated eigenvalues of A and $[\mu_1, \mu_2]$ the true values of the perturbed matrix $\tilde{A}$ then for each $A \in \mathbb{C}^{2\times 2}$ with $\frac{\sqrt{(\lambda_1 - \mu_1)^2 + (\lambda_2 - \mu_2)^2}}{\sqrt{(\mu_1)^2 + (\mu_2)^2}} = \mathcal{O}(\epsilon)$ there exists such $\tilde{A}$ where

$\frac{||\tilde{A} - A||}{||\tilde{A}||} = \mathcal{O}(\epsilon)$

3b

We know that this algorithm $\tilde{f}$ for f is backward stable if for each $A \in \mathbb{C}^{2\times 2}$, $\exists \tilde{A}$ such that (using the notation above) $\lambda_1 = \mu_1, \lambda_2 = \mu_2$ with $\frac{||\tilde{A} - A||}{||\tilde{A}||} = \mathcal{O}(\epsilon)$

3d

We know $\lambda = \frac{trace(A) \pm \sqrt{trace(A)^2 - 4det(A)}}{2} \Rightarrow r = \frac{\delta c \pm \sqrt{(\delta c)^2 - 4\delta c}}{2} \Rightarrow \delta r = \delta c \pm \sqrt{(\delta c)^2 - 4\delta c} \Rightarrow \frac{|\delta r / r|}{|\delta c / c|} = \frac{|\delta r|}{|\delta c|} = \frac{|\delta c \pm \sqrt{(\delta c)^2 - 4\delta c}|}{|\delta c|} \approx |1 \pm \sqrt{1 \pm 4(\delta c)^-}$
so our expected scaling of this error with machine epsilon is roughly one.

Question 4

–

4a

Using the code in cw2/coursework2q4.py we create a few different A matrices of varying sizes and when we inspect the LU decomposition we see that both L and U have the same block like structure as A but the blocks are cut diagonally to fit the lower and upper triangle criteria accordingly. So the only non negative parts of L are in the lower triangular blocks of $L_{1:5,1:5}, L_{5:9,5:9}\ldots, L_{4(n-1)+1:4(n+1),4(n-1)+1:4(n+1)}$ and it is the exact same for U but with the blocks being upper triangular. When decomposing a general matrix with the same bandwidths as A we obtain L with the same lower bandwidth as A and U with the same upper bandwidth as A. This means that for this specific A we have greater sparsity since inside the bandwidths of L and U we have more zeros than for a general matrix of same bandwidths as A.

4b

For banded LU we moved down and across each vector the same distance based upon p and q for each loop. Now we do not need to do this since we include operating with zeros unnecessarily. We only need to loop down and across until the end of our block. So when looping over $k = 1, 2, 3, 4$ our vectors that we update are up to row 5, $k = 5, 6, 7, 8$ our vectors that we update are k to 9, $k = 9, 10, 11, 12$ our vectors that we update are up to row 13 and so on. We can define $n = 5 + 4\lfloor\frac{k-1}{4}\rfloor$ to give the index of the row or column up to which we operate on for each vector updated. Hence we may write the algorithm as follows:\ $\cdot\ U \leftarrow A$ \
$\cdot\ L \leftarrow I$\ $ind = [5, 5, 5, 5, 9, 9, 9, 9, 13, 13, 13, 13, \ldots, m, m, m, m]$\ FOR k=1 TO m−1\ $z = ind[k]$

```
    FOR j=k+1 TO z
        $l_{jk} \leftarrow \frac{u_{jk}}{u_{kk}}$
        $u_{j,k:z} \leftarrow u_{j,k:z} - l_{jk}u_{k,k:z}$\    END FOR
END FOR\
```

We can computatioanlly initialise ind using $ind[k] = 5 + 4\lfloor\frac{k-1}{4}\rfloor$ FOR k=1 TO m−1\ Of course this is equivalent to decomposing each individual block into LU inplace. So we have a refined method since it works with less looping also in cw2/coursework2q4.py which works by using the LU_inplace method from the weekly exercises as follows.\ FOR $k = 1$ TO $(m-1)/4$\
LU*inplace($A*{4k:4k+5,4k:4k+5}$)\

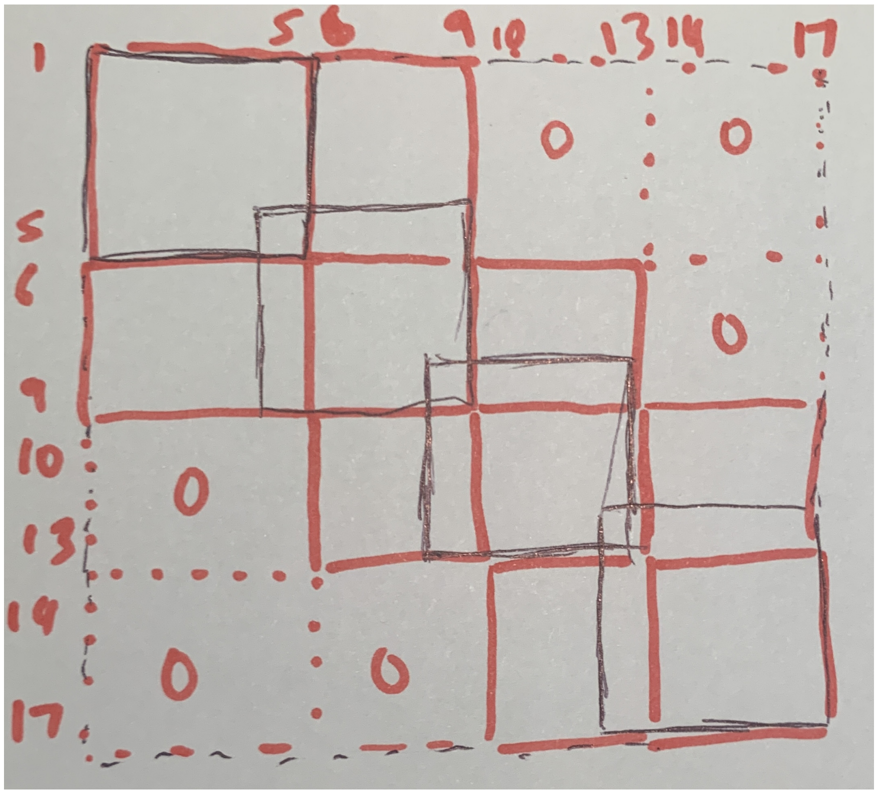
In each LU*inplace for the U update we have\ $N*{FLOPS} = \sum*{k=1}^{m-1}\sum*{j=k+1}^{m}2(m-k+1)=\sum*{k=1}^{m-1}2(m-k+1)\sum*{j=k+1}^{m}1 = 2m^2(m-1)-4m\frac{m(m-1)}{2}+2\frac{(m-1)m(2m-1)}{6}=\frac{(m-1)m(2m-1)}{3}*where*m *isthesizeoftheexactmatrixweareperformingLUonsointhiscaseitis*5*.Thereforeperblockwehave*60*FLOPSforUupdate. Thenwehaveanadditional*4 + 3
m\times m*wehaving*\frac{m-1}{4}*overlappingblocks. HencethetotalFLOPSforthemodifiedLUis*70\frac{m-1}{4}= 35\frac{m-1}{2} \sim \mathcal{O}(m). *Weexpectedthistobeatleastlinearin*m*sincefromthenotesweknowthatthebandedLUalgorithmisalsolinearin*m$ and in this algorithm we have less FLOPS by the structure of A.

4c

Code and pytest in cw2/coursework2q4.py

4d

We seek to convert our A into a tridiagonal block matrix where entries will be matrices. We illustrate the idea with the following diagram. In the diagram we see in black that there are 4 overlapping 5×5 blocks and the rest of the matrix is just zeros. By dividing the matrix into the red blocks as pictured we obtain a tridiagonal matrix of matrices.



The algorithm will be first to divide our matrix in the above way to get the tridiagonal structure.\ FOR $i = 1$ to $\frac{m-1}{4}$\ FOR $j = 1$ to $\frac{m-1}{4}$\ IF $i = j = 1$\ $\tilde{A}_{ij} = A_{1:5,1:5}$\ IF $i = 1, j \neq 1$\ $\tilde{A}_{ij} = A_{1:5,4(j-1)+2:4j+1}$\ IF $i \neq 1, j = 1$\

Processing math: 83%

$\tilde{A}_{ij} = A_{4(i-1)+2:4i+1,1:5}$\ ELSE\ $\tilde{A}_{ij} = A_{4(i-1)+2:4i+1,4(j-1)+2:4j+1}$\ If the original matrix vector equation was $A\begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_m \end{bmatrix} = \begin{bmatrix} b_1 \\ b_2 \\ \vdots \\ b_m \end{bmatrix}$ we transform it into the equation $\tilde{A}\begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_t \end{bmatrix} = \begin{bmatrix} c_1 \\ c_2 \\ \vdots \\ c_t \end{bmatrix}$

where $y_1 = \begin{bmatrix} x_1 \\ \vdots \\ x_5 \end{bmatrix}$ and for $i \neq 1$ we have $y_i = \begin{bmatrix} x_{4(i-1)+2} \\ \vdots \\ x_{4i+1} \end{bmatrix}$ and where $c_1 = \begin{bmatrix} b_1 \\ \vdots \\ b_5 \end{bmatrix}$ and for $i \neq 1$ we have $c_i = \begin{bmatrix} b_{4(i-1)+2} \\ \vdots \\ b_{4i+1} \end{bmatrix}$ We solve the reduced system using the banded LU algorithm with $p = q = 1$ and then by using forward and then backward substitution to obtain the y values (which will be vectors) in the reduced system and then we can concatenate all the y vectors to return to the original x vector. There will be a few differences from the the notes provided algorithm for LU banded decomposition, forward and backward substituion. In LU banded, when setting the 1s on the diagonal we now set identity blocks instead since 1 is the scalar multiplication identity. In forward and backward substitution when dividing by a scalar we now multiply by the inverse of the matrix which we calculate using our inverse_LU function. Below we outline the changes in pseudocode.\ y[0] = c[0] for i in range(1,m): y[i][0] = c[i][0] - L[i,i-1] @ y[i-1][0] #implementing forward subst x[-1][0] = inverse_LU(Atilde[-1, -1]) @ y[-1][0] for i in range(m-2,-1,-1): x[i][0] = inverse_LU(Atilde[i,i]) @ (y[i][0] - Atilde[i, i+1] @ x[i+1][0]) #implementing backward subst\ When thinking about number of FLOPS for this solution we first think about the number of FLOPS per each algorithm for banded LU with p=q=1 we have N_{FLOPS} \sim \mathcal{O}(m). Further, the resulting matrix L has lower bandwidth p and U has upper bandwidth q. This means that we can also exploit this structure in the forward and back substitution algorithms as well. For example, the forward substitution algorithm also has N_{FLOPS} \sim \mathcal{O}(m) which is the same as the backward substitution. Therefore the total number of flops for solving a system Ax=b using these three algorithms is also N_{FLOPS} \sim \mathcal{O}(m) since the sum of linear complexity will also be linear. But in my algorithm we multiplying by matrices and calculating inverse instead of multiplying by scalars and dividing by scalars. However since these matrices and inverses are no larger than 5 \times 5 and do not depend on m they will only contribute by the total FLOPS being a larger multiple of what it would be for a matrix of scalars. Hence, for my algorithm this does not actually affect the operation count and so my algorithm has N_{FLOPS} \sim \mathcal{O}(m). Therefore we get \alpha=1.

4e

Code and pytest in cw2/coursework2q4.py

\underline{\text{Question 5}}

5a

First of all we begin by looking at the structure of v. By definition v= [u_{1,1}, \dots u_{1,n-1}, u_{2,1}, \dots u_{2,n-1} \dots u_{n-1,1}, \dots u_{n-1,n-1}]^T We may use this vector as a guide in defining our A matrix. We observe from the equation that the structure of A will consist of 5 diagonals which are the (i,j),(i,j+1),(i,j-1) which all lie next to each other in a tridiagonal fashion along the centre of A and then the (i+1,j),(i-1,j) diagonals which only come in to play with a bandwidth of n-1 above and below respectively. So we can define A as follows:\ A_{i,i} = \frac{4\mu}{\Delta x^2} + c\ A_{i,i+1} = \frac{b^2_{i,i}}{2\Delta x} - \frac{\mu}{\Delta x^2}\ A_{i,i-1} = -\frac{b^2_{i,i}}{2\Delta x} - \frac{\mu}{\Delta x^2}\ A_{i,i+n-1} = \frac{b^1_{i,j}}{2\Delta x} - \frac{\mu}{\Delta x^2}\ A_{i-n+1,i} = -\frac{b^1_{i,i}}{2\Delta x} - \frac{\mu}{\Delta x^2}\ for all i such that that point would be inside A and 0 elsewhere\ And S = [S_{1,1}, \dots S_{1,n-1}, S_{2,1}, \dots S_{2,n-1} \dots S_{n-1,1}, \dots S_{n-1,n-1}]^T.\ To give us Av=S.\ For A we have a upper and lower bandwidth of n-1 and the matrix size of A is (n-1)^2\times(n-1)^2 so using the operation count from the notes for the banded LU algorithm we have N_{FLOPS} \sim \mathcal{O}((n-1)^4) = \mathcal{O}(n^4)

5b

The code implementation can be found in cw2/coursework2q5.py. \ We show that the operation count is \sim \mathcal{O}(mpq)\ N_{FLOPS} = \sum_{k=1}^{m-1}\sum_{j=k+1}^{k+p}(1+2q)=(1+2q)\sum_{k=1}^{m-1}\sum_{j=k+1}^{k+p}1 = (1+2q)\sum_{k=1}^{m-1}p = (1+2q)p(m-1) \sim \mathcal{O}(mpq) Further we show this is the correct order in cw2/coursework2q5.py. by testing counting the FLOPS for various banded matrices.

In [ ]: