Computational Linear Agebra: Exercises 1

Peter R. Taylor

1. Prove the Cauchy-Schwarz inequality
$$|\mathbf{x}^T\mathbf{y}| \leq \|\mathbf{x}\|_2 \, \|\mathbf{y}\|_2.$$
(Hint: consider a vector of the form $\mathbf{r} + \alpha\mathbf{s}$ and expand the scalar product.)

2. We can define a new type of product, denoted by $\circ$, for 2-vectors as
$$\begin{pmatrix} a \\ b \end{pmatrix} \circ \begin{pmatrix} c \\ d \end{pmatrix} = \begin{pmatrix} ac - bd \\ ad + bc \end{pmatrix}.$$
   (a) Show that this provides a vector formulation of multiplication of two complex numbers $a + b\mathrm{i}$ and $c + d\mathrm{i}$.

   (b) How does this influence the computational cost of complex compared to real arithmetic? How could a chip designer improve this?

   (c) Your CPU handles 64-bit arithmetic, but you need to do 128-bit "quadruple precision" multiplication (say). Modify the two-vector product operation above to produce the desired result. Again, what is the computational cost?

   (d) Now, consider multiplication of *quaternions* $a + b\mathrm{i} + c\mathrm{j} + d\mathrm{k}$, where the quaternion units multiply as
$$\mathrm{i}^2 = \mathrm{j}^2 = \mathrm{k}^2 = -1, \quad \mathrm{ij} = \mathrm{k} = -\mathrm{ji}, \quad \mathrm{jk} = \mathrm{i} = -\mathrm{kj}, \quad \mathrm{ki} = \mathrm{j} = -\mathrm{ik}.$$
   Define an appropriate rule for multiplication of 4-vectors that provides a vector formulation of quaternion multiplication. Confirm that quaternion multiplication is not commutative.

3. The general formula for matrix multiplication was given in the lecture. Three matrix indices were involved, so this formula implies three nested loops. There are six permutations of three objects.

   (a) Write down all six possible loop structures.

   (b) What type of vector operation is implied by the innermost loop in each case?

   (c) What are the implications for memory access in each case, and how might this affect performance?

4. Consider now the matrix triple product $\mathbf{C}{=}\mathbf{B}^T\mathbf{A}\mathbf{B}$, where T denotes a matrix transpose. In terms of explicit summations we can write
$$C_{ij} = \sum_k \sum_l B_{ki} A_{kl} B_{lj}.$$

(a) What is the operation count (or scaling with matrix dimension N) of this expression?

(b) How can we reduce the operation count? What are the consequences for storage?

(c) How would you code this if you could use at most an extra array of length N as an intermediate? If the matrix $\mathbf{A}$ could be overwritten (and all matrices are square)?

(d) Now consider the case in which $\mathbf{A}$ and $\mathbf{B}$ are block-diagonal with the following structure

$$\begin{pmatrix} \mathbf{A}(N/2, N/2) & \mathbf{0} \\ \mathbf{0} & \mathbf{A}(N/2, N/2) \end{pmatrix}$$

and similarly for $\mathbf{B}$. What is the operation count of the triple product? Deduce the factor by which the work is reduced when the matrices are block-diagonal in m blocks of the same dimension.

(e) In many physical applications, a triple product like $\mathbf{B}^T\mathbf{AB}$ is encountered in which the matrix $\mathbf{A}$ is symmetric ($A_{ij} = A_{ji}$). If we store only the nonzero elements of $\mathbf{A}$, using a one-dimensional index $k = i(i-1)/2 + j$, where $i = 1, N$ and $j = 1, i$, construct the inner loop of a program to calculate the matrix product. How might this affect performance compared to the use of full square matrices?