Pr. 1. (sol/hs001)

(a) Let $\boldsymbol{x} \in \mathbb{R}^M$ be a vector with $x_i = i$ for $i = 1, \dots, M$. Let $\boldsymbol{y} \in \mathbb{R}^N$ be a vector with $y_j = 2^j$ for $j = 1, \dots, N$. Then the desired $M \times N$ matrix is given by the outer-product $\boldsymbol{x}\boldsymbol{y}'$.

(b) In `Julia` (try it!):

```
M,N = 3,2
x = 1:M # this is a column vector in Julia!
y = 2 .^ (1:N) # element-wise exponentiation
A = x * y' # 3-line version is probably more clear
A = (1:M) * (2 .^ (1:N))' # one-line version per problem statement
```

Pr. 2. (sol/hs002)
If $\boldsymbol{Q} \in \mathbb{R}^{N \times N}$ denotes an orthogonal matrix, then $\boldsymbol{Q}^T\boldsymbol{Q} = \boldsymbol{I}$ or equivalently, $\det(\boldsymbol{Q}^T\boldsymbol{Q}) = \det(\boldsymbol{I}) = 1$. Since $\det(\boldsymbol{Q}^T\boldsymbol{Q}) = \det(\boldsymbol{Q}^T) \cdot \det(\boldsymbol{Q})$ and $\det(\boldsymbol{Q}^T) = \det(\boldsymbol{Q})$, we have that $\det(\boldsymbol{Q})^2 = 1$, or, equivalently, that $\det(\boldsymbol{Q}) = \pm 1$.
It is always instructive to check your answers programmatically.
In `Julia` type:

```
for n=1:9
    Q , _ = qr(randn(n, n)) # the "_" means ignore "R" output
    @show det(Q)
end
```

How does the answer change when $\boldsymbol{Q}$ is unitary? Type in the code

```
for n=1:9
    Q , _ = qr(randn(n, n) + 1im * randn(n, n))
    @show det(Q), abs(det(Q))
end
```

The answer follows a similar argument to show that when $\boldsymbol{Q} \in \mathbb{C}^{N \times N}$ is a unitary matrix, then $|\det(\boldsymbol{Q})| = 1$. The `qr` command invokes an algorithm equivalent to the **Gram-Schmidt procedure** to generate an orthogonal (or unitary) matrix from the matrix argument.

Pr. 3. (sol/hs004)

(a) Grader: the course notes provide a **matrix determinant lemma** that makes this problem trivial: $\det(\boldsymbol{A} + \boldsymbol{x}\boldsymbol{y}') = (1 + \boldsymbol{y}'\boldsymbol{A}^{-1}\boldsymbol{x}) \det(\boldsymbol{A})$. It is acceptable for students to use that lemma with $\boldsymbol{A} = \boldsymbol{I}$ to "solve" part (a).

Here is a solution that uses first principles.
In Hint 3, substitute $\boldsymbol{A} = 1, \boldsymbol{B} = \boldsymbol{y}', \boldsymbol{C} = \boldsymbol{x}$, and $\boldsymbol{D} = \boldsymbol{I}$. Then:

$$\det(\boldsymbol{I}_N - \boldsymbol{x}\boldsymbol{y}') = \det(1) \det(\boldsymbol{I}_N - \boldsymbol{x}1^{-1}\boldsymbol{y}') = \det\left(\begin{bmatrix} 1 & \boldsymbol{y}' \\ \boldsymbol{x} & \boldsymbol{I}_N \end{bmatrix}\right).$$

By Hint 4:

$$\det\left(\begin{bmatrix} 1 & \boldsymbol{y}' \\ \boldsymbol{x} & \boldsymbol{I}_N \end{bmatrix}\right) = \det(\boldsymbol{I}_N) \det(1 - \boldsymbol{y}'\boldsymbol{I}_N^{-1}\boldsymbol{x}) = \det(1 - \boldsymbol{y}'\boldsymbol{x}) = 1 - \boldsymbol{y}'\boldsymbol{x}.$$

Thus $\det(\boldsymbol{I}_N - \boldsymbol{x}\boldsymbol{y}') = 1 - \boldsymbol{y}'\boldsymbol{x}$. The key to solving this problem is recognizing that, with appropriate choices of $\boldsymbol{A}$, $\boldsymbol{B}$, $\boldsymbol{C}$ and $\boldsymbol{D}$, one can invoke the determinant properties in hints 3 and 4 of the problem to simplify the computation.

(b)

$$\det(\lambda \boldsymbol{I}_N - \boldsymbol{x}\boldsymbol{y}') = \det\left((\lambda \boldsymbol{I}_N)\left(\boldsymbol{I}_N - \frac{\boldsymbol{x}\boldsymbol{y}'}{\lambda}\right)\right) = \det(\lambda \boldsymbol{I}_N) \det(1 - (\boldsymbol{x}\boldsymbol{y}')/\lambda) = \lambda^N \left(1 - \frac{\boldsymbol{y}'\boldsymbol{x}}{\lambda}\right) \text{ (from part (a))}$$
$$= \lambda^{N-1}(\lambda - \boldsymbol{y}'\boldsymbol{x}).$$

The potential division by zero (when $\lambda = 0$) is a potential flaw in the above argument. Here's how to make it rigorous. For $N = 1$ the problem is trivial, so we focus on the cases where $N > 1$. First consider the case $\lambda = 0$. It is clear that $\det(-\boldsymbol{x}\boldsymbol{y}') = 0$ because $\boldsymbol{x}\boldsymbol{y}'$ has rank 1 and is thus singular (when $N > 1$). Thus $\lambda = 0$ is one eigenvalue of $\boldsymbol{x}\boldsymbol{y}'$. Now for the case where $\lambda \neq 0$, the derivation above that involves dividing by λ is fine, and we conclude that $\lambda = \boldsymbol{y}'\boldsymbol{x}$ is the only possibly nonzero eigenvalue. Thus all other eigenvalues must be zero, leading to the characteristic polynomial shown above.

(c) Setting the result of part (b) to 0, we find that $\lambda = \boldsymbol{y}'\boldsymbol{x}$ is one solution, and the root $\lambda = 0$ is repeated $N - 1$ times.

(d) Note that when $\boldsymbol{y}'\boldsymbol{x} = 0$ then all eigenvalues of the matrix $\boldsymbol{x}\boldsymbol{y}'$ are identically zero. For this to occur we just need $\boldsymbol{x}$ and $\boldsymbol{y}$ to be orthogonal to each other; in contrast the matrix $\boldsymbol{x}\boldsymbol{y}'$ will not equal the zero matrix unless $\boldsymbol{x}$ or $\boldsymbol{y}$ (or both) are zero! In that sense the eigenvalues are not always characteristic of the matrix even though eigenvalues mean "characteristic" values (in German)!
Note that $\boldsymbol{x}\boldsymbol{y}'$ times any vector is always a multiple of $\boldsymbol{x}$, *i.e.*, $\boldsymbol{x}\boldsymbol{y}'\boldsymbol{v} = (\boldsymbol{y}'\boldsymbol{v})\boldsymbol{x}$.
Thus the only eigenvector of the matrix $\boldsymbol{x}\boldsymbol{y}'$ is $\boldsymbol{x}$ (or scalar multiples thereof), but $(\boldsymbol{x}\boldsymbol{y}')\boldsymbol{x} = \boldsymbol{x}(\boldsymbol{y}'\boldsymbol{x}) = 0$ when $\boldsymbol{x}$ and $\boldsymbol{y}$ are orthogonal.

Pr. 4. (sol/hs005)

(a) Let $\boldsymbol{A}, \boldsymbol{B} \in \mathbb{F}^{N \times N}$. Consider scalars $\alpha, \beta \in \mathbb{F}$. Then:

$$\begin{aligned}\operatorname{Tr}(\alpha\boldsymbol{A} + \beta\boldsymbol{B}) &= \sum_{i=1}^{N} (\alpha\boldsymbol{A} + \beta\boldsymbol{B})_{ii} = \alpha \sum_{i=1}^{N} A_{ii} + \beta \sum_{i=1}^{N} B_{ii} \quad \text{(by linearity of matrix addition)} \\ &= \alpha \operatorname{Tr}(\boldsymbol{A}) + \beta \operatorname{Tr}(\boldsymbol{B}).\end{aligned}$$

(b) Even though $\boldsymbol{A}\boldsymbol{B}$ and $\boldsymbol{B}\boldsymbol{A}$ are square matrices, $\boldsymbol{A}\boldsymbol{B}$ and $\boldsymbol{B}\boldsymbol{A}$ need not have the same dimensions for the identity to hold. Let $\boldsymbol{A} \in \mathbb{F}^{N \times M}$ and $\boldsymbol{B} \in \mathbb{F}^{M \times N}$ so that $\boldsymbol{A}\boldsymbol{B} \in \mathbb{R}^{N \times N}$ and $\boldsymbol{B}\boldsymbol{A} \in \mathbb{R}^{M \times M}$.

When $M \neq N$ it is clear that $\boldsymbol{A}\boldsymbol{B} \neq \boldsymbol{B}\boldsymbol{A}$ from a purely dimension matching argument; in general $\boldsymbol{A}\boldsymbol{B} \neq \boldsymbol{B}\boldsymbol{A}$ even when $M = N$. (Try it out for some simple matrices in `Julia`.) $\boldsymbol{A}\boldsymbol{B} \neq \boldsymbol{B}\boldsymbol{A}$ for matrices is evidence of the fact that matrix multiplication is not commutative in general, unlike scalar multiplication where $ab = ba$ for scalar a and b. When $\boldsymbol{A}$ and $\boldsymbol{B}$ are diagonal then $\boldsymbol{A}\boldsymbol{B} = \boldsymbol{B}\boldsymbol{A}$, but this is clearly a very special and restricted class of matrices. (There are other classes that commute, such as circulant matrices.)

Now onto the problem at hand. Expressing $\boldsymbol{A}$ in row partition and $\boldsymbol{B}$ in column partition yields the expression:

$$\boldsymbol{A}\boldsymbol{B} = \begin{bmatrix} A_{1,:} \\ \dots \\ A_{N,:} \end{bmatrix} \begin{bmatrix} B_{:,1} \dots B_{:,N} \end{bmatrix},$$

so that the diagonal elements $(\boldsymbol{A}\boldsymbol{B})_{ii} = A_{i,:}B_{:,i}$ so that:

$$\operatorname{Tr}(\boldsymbol{A}\boldsymbol{B}) = \sum_{i=1}^{N} A_{i,:}B_{:,i} = \sum_{i=1}^{N} \sum_{j=1}^{M} A_{ij}B_{ji}.$$

Similarly, expressing $\boldsymbol{B}$ in row partition and $\boldsymbol{A}$ in column partition yields the expression:

$$\boldsymbol{B}\boldsymbol{A} = \begin{bmatrix} B_{1,:} \\ \dots \\ B_{M,:} \end{bmatrix} \begin{bmatrix} A_{:,1} \dots A_{:,M} \end{bmatrix},$$

so that the diagonal entries are $(\boldsymbol{B}\boldsymbol{A})_{jj} = B_{j,:}A_{:,j}$, so that:

$$\operatorname{Tr}(\boldsymbol{B}\boldsymbol{A}) = \sum_{j=1}^{M} B_{j,:}A_{:,j} = \sum_{j=1}^{M} \sum_{i=1}^{N} A_{ij}B_{ji} = \operatorname{Tr}(\boldsymbol{A}\boldsymbol{B}).$$

A more elegant way of "seeing" this answer is to keep $\boldsymbol{B}$ in column partition and $\boldsymbol{A}$ in row partition and focus on the diagonal elements of the matrix product $\boldsymbol{BA}$. Recall that

$$\boldsymbol{BA} = \begin{bmatrix} B_{:,1} \dots B_{:,N} \end{bmatrix} \begin{bmatrix} A_{1,:} \\ \dots \\ A_{N,:} \end{bmatrix},$$

which can be expanded in terms of outer-products to yield

$$\boldsymbol{BA} = \sum_{i=1}^{N} B_{:,i} A_{i,:}.$$

Part (a) established that the trace of the sum of matrices is the sum of the traces. The desired result follows by noting that for every i, the sum of the diagonal elements of the matrix $B_{:,i}A_{i,:}$ identically equals $A_{i,:}B_{:,i}$.

(c) When $\boldsymbol{S}$ is skew-symmetric, $\boldsymbol{S}^T = -\boldsymbol{S}$. Thus the diagonal entries must satisfy the relation $S_{ii} = (\boldsymbol{S}^T)_{ii} = (-\boldsymbol{S})_{ii} = -S_{ii} \Rightarrow S_{ii} = 0$. Since $\mathrm{Tr}(\boldsymbol{S}) = \sum_i S_{ii} \Rightarrow \mathrm{Tr}(\boldsymbol{S}) = 0$.

(d) An example of $\boldsymbol{S}$ that has $\mathrm{Tr}(\boldsymbol{S}) = 0$ but is not skew-symmetric is: $\boldsymbol{S} = \begin{bmatrix} 0 & -1 \\ 3 & 0 \end{bmatrix}$.

(e) This can be shown by direct multiplication and the trig identity $\sin^2(\theta) + \cos^2(\theta) = 1$:

$$\begin{aligned}
\boldsymbol{A}^2 &= \frac{1}{4} \begin{bmatrix} 2\cos^2(\theta) & \sin(2\theta) \\ \sin(2\theta) & 2\sin^2(\theta) \end{bmatrix} \begin{bmatrix} 2\cos^2(\theta) & \sin(2\theta) \\ \sin(2\theta) & 2\sin^2(\theta) \end{bmatrix} \\
&= \frac{1}{4} \begin{bmatrix} 4\cos^4(\theta) + \sin^2(2\theta) & 2\cos^2(\theta)\sin(2\theta) + \sin(2\theta)2\sin^2(\theta) \\ 2\cos^2(\theta)\sin(2\theta) + \sin(2\theta)2\sin^2(\theta) & \sin^2(2\theta) + 4\sin^4(\theta) \end{bmatrix} \\
&= \frac{1}{4} \begin{bmatrix} 4\cos^4(\theta) + 4\sin^2(\theta)\cos^2(\theta) & 2\cos^2(\theta)\sin(2\theta) + \sin(2\theta)2\sin^2(\theta) \\ 2\cos^2(\theta)\sin(2\theta) + \sin(2\theta)2\sin^2(\theta) & 4\sin^2(\theta)\cos^2(\theta) + 4\sin^4(\theta) \end{bmatrix} \\
&= \frac{1}{2} \begin{bmatrix} 2\cos^2(\theta) & \sin(2\theta) \\ \sin(2\theta) & 2\sin^2(\theta) \end{bmatrix}.
\end{aligned}$$

A tidier solution is to use $\sin(2\theta) = 2\sin(\theta)\cos(\theta)$ and write $\boldsymbol{A} = \boldsymbol{v}\boldsymbol{v}'$ where $\boldsymbol{v} = \begin{bmatrix} \cos\theta \\ \sin\theta \end{bmatrix}$, and then recognize that $\boldsymbol{v}'\boldsymbol{v} = 1$.

Pr. 5. (sol/hsj01)
The solutions are all in your notebook.
Grader: give full credit if the pdf shows that the student reached the end of the tutorial, otherwise 0 points.

Pr. 6. (sol/hs007)

(a) We solve for λ in $\det(\boldsymbol{A} - \lambda \boldsymbol{I}) = 0$ as follows:

$$\det\left(\begin{bmatrix} 6-\lambda & 16 \\ -1 & -4-\lambda \end{bmatrix}\right) = 0 \Rightarrow (6-\lambda)(-4-\lambda) + 16 = 0 \Rightarrow \lambda^2 - 2\lambda - 8 = 0 \Rightarrow \lambda = \frac{2 \pm \sqrt{4+32}}{2} = 4, -2$$

(b) $\det(\boldsymbol{A}) = -8 =$ Product of the eigenvalues.

$\text{Tr}(\boldsymbol{A}) = 2 =$ Sum of the eigenvalues.

This product and sum property holds for any square matrix $\boldsymbol{A}$.

(c) `using LinearAlgebra: eigen`

`A = [6 16; -1 -4]; _,V = eigen(A); V'V`

produces the following output showing the eigenvectors are *not* orthogonal, because this matrix $\boldsymbol{A}$ is not a normal matrix:

```
2×2 Array{Float64,2}:
  1.0       -0.94299
 -0.94299    1.0
```

(d) (not graded)

Most high-level languages, like `MATLAB`, `Python`, and `Julia`, call **LAPACK** to compute eigenvalues, and may or may not sort the results. Eigenvalues are complex in general so there is no canonical way to order them. The important thing to realize is that the order may not be what you want!

Suppose you want to sort complex eigenvalues in increasing order of absolute value. You can write your own complex sort easily.

This can be implemented in `Julia` as

```
compare(v,w) = abs(v) < abs(w) # Use > to sort other way
csort(x) = sort(x, lt=compare)
abs.(csort(eigvals(randn(100,100)))) # apply abs element-wise
```

`Julia` 1.1 (and earlier versions) provide no eigenvalue sorting by default. Starting with `Julia` 1.2, the eigenvalues of general matrices are sorted in *increasing* order (real part first) by default. Functions like `eigen` and `eigvals` have options to override this behavior. However, for special matrices like `Diagonal` the eigenvalues are not sorted! So it is probably safest to assume that eigenvalues are unsorted for portability of code between different languages and different `Julia` versions.

Pr. 7. (sol/hs008)

Let $\boldsymbol{x} = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix}$ and $\boldsymbol{A} = \begin{bmatrix} A_{11} & \dots & A_{1n} \\ & \ddots & \\ A_{n1} & \dots & A_{nn} \end{bmatrix}$. Then $[\boldsymbol{A}\boldsymbol{x}]_i = \sum_{j=1}^n A_{ij} x_j$, so

$$y = \boldsymbol{x}' \boldsymbol{A} \boldsymbol{x} = \sum_{i=1}^n x_i^* \, [\boldsymbol{A}\boldsymbol{x}]_i = \sum_{i=1}^n x_i^* \left(\sum_{j=1}^n A_{ij} x_j \right) = \sum_{i=1}^n \sum_{j=1}^n x_i^* A_{ij} x_j.$$

Pr. 8. (sol/hs009)
Let the columns of $\boldsymbol{U}$ be $\{\boldsymbol{u}_k\}_{k=1}^M$ and the columns of $\boldsymbol{V}$ be $\{\boldsymbol{v}_k\}_{k=1}^N$.

(a) When $M < N$:

$$\boldsymbol{A} = \sum_{k=1}^{M} \sigma_k \boldsymbol{u}_k \boldsymbol{v}_k'.$$

Therefore, M outer products are summed together.
An even "more efficient" answer is that it is the number of nonzero σ_k values.

(b) When $M > N$:

$$\boldsymbol{A} = \sum_{k=1}^{N} \sigma_k \boldsymbol{u}_k \boldsymbol{v}_k'.$$

Therefore, N outer products are summed together, again in the typical case where all the σ_k values are nonzero.

(c) In general, we can write $\boldsymbol{A}$ in outer-product form using (at most) $\min(M, N)$ outer products as

$$\boldsymbol{A} = \sum_{k=1}^{\min(M,N)} \sigma_k \boldsymbol{u}_k \boldsymbol{v}_k'.$$

This property is important for the **SVD**.

Pr. 9. (sol/hs003jf)
This did not seem to cause confusion, but in one-based-indexing languages such as `MATLAB` or `Julia`, we might have used the variation

$$y[m'] = \sum_{k=-\infty}^{\infty} h[k]\, x[m'-k+1] = \sum_{k=-\infty}^{\infty} h[m'-k+1]\, x[k], \; m' = 1, \ldots, M.$$

The equality $\sum_{k=-\infty}^{\infty} h[k]\, x[m'-k+1] = \sum_{k=-\infty}^{\infty} h[m'-k+1]\, x[k]$, the commutativity property of convolution, can be understood by realizing that either way $y[m']$ is a sum of all x values and h values whose indices add to $m'+1$.
Sometimes this is written

$$y[m'] = \sum_{\alpha+\beta=m'+1} x[\alpha]\, y[\beta],$$

to emphasize the symmetry. Returning to the problem at hand, we would like an $\boldsymbol{H}$ matrix that expresses matrix times vector multiplication:

$$y_{m'} = \sum_{n'} H_{m'n'}\, x_{n'} = \sum_{n'=1}^{N} h[m'-n'+1]\, x[n'], \; m' = 1, \ldots, M.$$

Thus it is easy to see that

$$H_{m'n'} = \begin{cases} h[m'-n'+1], & 1 \le m'-n'+1 \le K \\ 0, & \text{else.} \end{cases}$$

The matrix $\boldsymbol{H}$ looks like this

$$\boldsymbol{H} = \begin{bmatrix} h[1] & 0 & 0 & \ldots & 0 \\ h[2] & h[1] & 0 & \ldots & 0 \\ h[3] & h[2] & h[1] & \ldots & 0 \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ 0 & 0 & 0 & 0 & h[K] \end{bmatrix}.$$

A possible `Julia` implementation is

```
"""
`H, y = convolution(h, x)`

Compute discrete convolution of the input vectors via
matrix multiplication, returning both the matrix `H` and result `y`

In:
- `h` vector of length `K`
- `x` vector of length `N`

Out:
- `H` `M × N` convolution matrix defined by `h`
- `y` vector of length `M` containing the discrete convolution of `h` and `x` computed using `H`.
"""
function convolution(h, x)

    # Parse inputs
    h = vec(h) # we're so nice...
    x = vec(x)
    K = length(h)
    N = length(x)

    # Construct convolution matrix
    M = N + K - 1 # here is the answer to the question about size "M"
    T = promote_type(eltype(h), eltype(x)) # professional version!
    H = zeros(T, M, N)
    for n in 1:N
        H[n:(n + K - 1), n] = h
    end

    y = H * x # Compute convolution via matrix multiplication

    return H, y
end
```

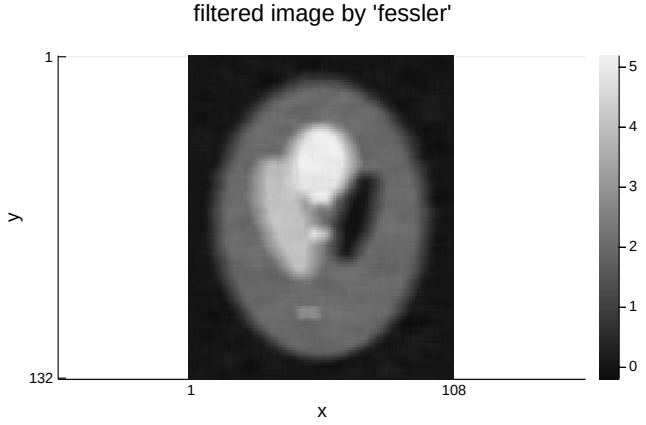
In `Julia`, we can construct $\boldsymbol{H}$ elegantly using the ternary operator `a ? b : c` and array comprehensions as

```
H = [(1 <= m-n+1 <= K) ? h[m-n+1] : 0 for m=1:(N+K-1), n=1:N]
```

which reads as "if $1 \leq m - n + 1 \leq K$, set $H[m, n] = h[m - n + 1]$, otherwise set $H[m, n] = 0$."
Thus an alternative solution is

```
function convolution(h,x)
    K, N = length(h), length(x)
    H = [(1 <= m-n+1 <= K) ? h[m-n+1] : 0 for m=1:(N+K-1), n=1:N]
    return H, H * x
end
```

Pr. 10. (sol/hs003jf-fig)
Here is a representative output image. Convolution with `ones(m)/m` is called a moving average filter and has the benefit of reducing noise by averaging, but also blurs out the signal because it is a low-pass filter.



Grader: if the submitted image says "by fessler" in the title or has no title at all then no credit for this problem.

Pr. 11. (sol/hsj12)

(a) Full credit answer:
`z*((u'*v)*(x'*y))`

Partial credit answers:
`(z*(u'*v))*(x'*y)`
`z*(u'*v)*(x'*y)`

Any other answer is no credit because it potentially require $O(N^2)$ multiplies.

(b) Analyzing the best answer: `z*((u'*v)*(x'*y))`

`tmp1 = (u'*v)` needs N
`tmp2 = (x'*y)` needs N
`tmp3 = tmp1*tmp2` needs 1
`z * tmp3` needs N

So the total is $3N + 1$ multiplies.

(Grader: assign 2/3 points for the answer $3N$. Any other answer is incorrect in general.)

Grader: if a student gave the answer `(z*(u'*v))*(x'*y)` for part (a), then the correct analysis for part (b) is $4N$ multiplies and they can earn full credit for part (b) in that case even though their answer to (a) was partial credit.

But if the student says `z*(u'*v)*(x'*y)` for part (a), and claims $3N$ or $3N+1$ then assign 1/3 points for part (b) because the compiler might choose to associate it the way that requires $4N$ multiplies.

Pr. 12. (sol/hs006u)
Given: $\boldsymbol{U}_i'\boldsymbol{U}_i = \boldsymbol{I}, \quad i = 1, 2, \ldots, k$, and let $\boldsymbol{U} = \boldsymbol{U}_1 \cdot \boldsymbol{U}_2 \cdots \boldsymbol{U}_k$. Now examine $\boldsymbol{U}'\boldsymbol{U}$:

$$\begin{aligned}
\boldsymbol{U}'\boldsymbol{U} &= (\boldsymbol{U}_1 \cdot \boldsymbol{U}_2 \cdots \boldsymbol{U}_k)' \cdot (\boldsymbol{U}_1 \cdot \boldsymbol{U}_2 \cdots \boldsymbol{U}_k) \\
&= \boldsymbol{U}_k' \boldsymbol{U}_{k-1}' \cdots \underbrace{\boldsymbol{U}_1' \boldsymbol{U}_1}_{=\boldsymbol{I}} \cdot \boldsymbol{U}_2 \cdots \boldsymbol{U}_k \\
&= \boldsymbol{U}_k' \cdot \boldsymbol{U}_{k-1}' \cdots \underbrace{\boldsymbol{U}_2' \cdot \boldsymbol{I} \cdot \boldsymbol{U}_2}_{=\boldsymbol{I}} \cdot \boldsymbol{U}_2 \cdots \boldsymbol{U}_k = \cdots = \boldsymbol{I},
\end{aligned}$$

showing that the square matrix $\boldsymbol{U} = \boldsymbol{U}_1 \cdot \boldsymbol{U}_2 \cdots \boldsymbol{U}_k$ is **unitary**.

Non-graded problem(s) below

Pr. 13. (sol/hsj11)

(a)
$$\begin{bmatrix} \boldsymbol{A} & \boldsymbol{B} & \boldsymbol{C} \end{bmatrix} \begin{bmatrix} \boldsymbol{D} \\ \boldsymbol{E} \\ \boldsymbol{F} \end{bmatrix} = \boldsymbol{AD} + \boldsymbol{BE} + \boldsymbol{CF}$$

(b)
$$\begin{bmatrix} \boldsymbol{A} \\ \boldsymbol{B} \end{bmatrix} \begin{bmatrix} \boldsymbol{C} & \boldsymbol{D} \end{bmatrix} = \begin{bmatrix} \boldsymbol{AC} & \boldsymbol{AD} \\ \boldsymbol{BC} & \boldsymbol{BD} \end{bmatrix}$$

(c)
$$\boldsymbol{A} \begin{bmatrix} \boldsymbol{B} & \boldsymbol{C} & \boldsymbol{D} \end{bmatrix} = \begin{bmatrix} \boldsymbol{AB} & \boldsymbol{AC} & \boldsymbol{AD} \end{bmatrix}$$

(d)
$$\begin{bmatrix} \boldsymbol{A} \\ \boldsymbol{B} \\ \boldsymbol{C} \end{bmatrix} \boldsymbol{D} = \begin{bmatrix} \boldsymbol{AD} \\ \boldsymbol{BD} \\ \boldsymbol{CD} \end{bmatrix}$$

(e)
$$\begin{bmatrix} \boldsymbol{A} & \boldsymbol{B} \\ \boldsymbol{C} & \boldsymbol{D} \end{bmatrix} \begin{bmatrix} \boldsymbol{E} & \boldsymbol{F} \\ \boldsymbol{G} & \boldsymbol{H} \end{bmatrix} = \begin{bmatrix} \boldsymbol{AE} + \boldsymbol{BG} & \boldsymbol{AF} + \boldsymbol{BH} \\ \boldsymbol{CE} + \boldsymbol{DG} & \boldsymbol{CF} + \boldsymbol{DH} \end{bmatrix}$$

(f)
$$(\boldsymbol{I}_2 \otimes \boldsymbol{A}) \begin{bmatrix} \boldsymbol{E} & \boldsymbol{F} \\ \boldsymbol{G} & \boldsymbol{H} \end{bmatrix} = \begin{bmatrix} \boldsymbol{AE} & \boldsymbol{AF} \\ \boldsymbol{AG} & \boldsymbol{AH} \end{bmatrix}$$

(g)
$$\begin{bmatrix} \boldsymbol{A} & \boldsymbol{B} \end{bmatrix} \begin{bmatrix} \boldsymbol{C} & \boldsymbol{0} \\ \boldsymbol{0} & \boldsymbol{D} \end{bmatrix} \begin{bmatrix} \boldsymbol{E} & \boldsymbol{F} \end{bmatrix}' = \boldsymbol{ACE}' + \boldsymbol{BDF}'$$

(h)
$$\operatorname{trace}\left(\begin{bmatrix} \boldsymbol{A} & \boldsymbol{B} \\ \boldsymbol{C} & \boldsymbol{D} \end{bmatrix} \right) = \operatorname{trace}(\boldsymbol{A}) + \operatorname{trace}(\boldsymbol{D})$$

(i)
$$\det\left(\begin{bmatrix} \boldsymbol{A} & \boldsymbol{0} \\ \boldsymbol{B} & \boldsymbol{C} \end{bmatrix} \right) = \det(\boldsymbol{A}) \det(\boldsymbol{C})$$

(j)
$$\boldsymbol{A}\operatorname{Diag}(\lambda_1, \ldots, \lambda_N) = \begin{bmatrix} \lambda_1 \boldsymbol{A}_{:,1}, \ldots \lambda_N \boldsymbol{A}_{:,N} \end{bmatrix}$$