4 Vectors

MATLAB was built around vectors and matrices. A vector is entered by enclosing values in square brackets (separated by spaces and optionally commas). For example
\[ A = [1 \ 6 \ 3 \ 5 \ 4 \ 9 \ 2] \quad \text{or} \quad A = [1, 6, 3]\]. The expression \( A(2) \) gives the second element in array \( A \): in the above examples it is 6.

There are several inbuilt vector functions e.g. \texttt{median} or \texttt{max}. Finding the maximum of an array is a standard programming example; here is the idea without exploiting the features of MATLAB:

\[
\text{max} = A(1);
\text{for } i = 2: \text{length}(A)
    \text{if } A(i) > \text{max}
        \text{max} = A(i);
    \end{\text{if}}
\end{\text{for}}
\]

One can readily build vectors. For example, expression 1:5 is the vector \([1 \ 2 \ 3 \ 4 \ 5]\) and expression 1:2:9 is the vector \([1 \ 3 \ 5 \ 7 \ 9]\). A useful feature is to apply a function \texttt{entry-wise} on an array. This uses a dot before the operator: e.g. the expression \( A \cdot 2 \) yields the array with each entry squared. (The operation \( A^2 \) attempts to square \( A \) as a matrix.) Note that a scalar function applied to a vector automatically operates entry-wise; e.g. \( \sin(0:0.1:2\pi) \) gives an array of values of \( \sin x \) for \( x \) running from 0 to \( 2\pi \) in increments of 0.1.

We can use this to approximate \( \pi \) again. The idea this time is that the area under the semicircle \( y = \sqrt{1 - x^2} \) in the first quadrant is \( \pi/4 \). The area is equal to the base times the average height (and the base has length 1). So \( \pi \) is approximated by 4 times an approximation to the average of \( y \) over the interval \( 0 \leq x \leq 1 \). We find the approximation by taking 1000 equally spaced values of \( x \).

\[
x = \text{linspace}(0, 1, 1000);
y = \text{sqrt}(1 - x \cdot 2);
\text{PIE} = 4 * \text{mean}(y)
\]

If one compares two vectors for equality with an \texttt{if} statement such as \texttt{if A==B}, it works. But if one enters \texttt{A==B} as a statement, one gets a vector: it is 1 where \( A \) and
B agree, and 0 where A and B disagree (and we get an error if A and B don’t have the same dimensions). It is suggested that you use isEqual(A,B) in the if statement.

But we can make use of logical vectors. For a logical vector X, the function all tests if all the entries true (or nonzero), while any tests if there is at least one that is true (or nonzero). For example, sum(A==B) gives the number of places the vectors A and B agree, and any(C>10) is true if and only if some entry in C is more than 10.

---

**Sample code**

Here is code to check whether all the entries in vector X are different or not.

```matlab
isDistinct=true;
for i=1:length(X)
    for j=i+1:length(X)
        if X(i)==X(j)
            isDistinct=false;
        end
    end
end
display(isDistinct);
```

---