

Motivating Example 1

You have N 3 element vectors \mathbf{x}_i given as a $3 \times N$ matrix X

You want to form a matrix of N , 3 element vectors:

$$\mathbf{y}_i = \mathbf{x}_i' * \Sigma^{-1} \mathbf{x}_i$$

As a $1 \times N$ matrix Y ;

Motivating Example 2

You have N 128 element vectors $X_1(:, i)$ given as a $128 \times N$ matrix X_1 .

Also a 128 dimensional vector \mathbf{x}_2 .

You want to form a N dimensional vector Y as:

The squareroot of the sum of square differences of each of the N X_1 vectors from the single \mathbf{x}_2 vector.

Vectorized Processing in MATLAB

- Basic Code Optimizations e.g. Pre Allocation.
a = zeros(n, 2);
- Avoid For Loops!
- Loops over particles will be extremely slow!
- Vectorized Processing (explained next)

- 2D matrices:

$$\text{size}(A) = [d \ n] \rightarrow A(i,j) = A(i + (j - 1) * d)$$

- n-D matrices:

$$\text{size}(A) = [d_1 \ d_2 \ \dots \ d_n] \rightarrow$$

$$A(i_1, i_2, \dots, i_n) = A(i_1 + (i_2 - 1)d_1 + (i_3 - 1)d_1d_2 + (i_4 - 1)d_1d_2d_3 + \dots + (i_n - 1)d_1\dots d_{n-1})$$

- Vectors better (should!) be stored column wise in matrices:
 $V_1, V_2, \dots, V_n \rightarrow V = [V_1 \ V_2 \ \dots \ V_n]$
 $V(i,j) \rightarrow$ i th dimension of the j th vector
 $size(V_1) = \dots = size(V_n) = [d \ 1] \rightarrow size(V) = [d \ n]$
- DIMENSIONS! The simplest and most efficient way to find out if something is wrong!

- Multiple entities(e.g. Matrices) are better stacked on the last singleton dimension:

$$E_1, E_2, \dots, E_n \rightarrow$$

$$E = \text{cat}(\text{ndims}(E_1) + 1, E_1, E_2, \dots, E_n)$$

$$\text{size}(E_1) = \dots = \text{size}(E_n) = [d_1 \ d_2 \ \dots \ d_D] \rightarrow$$

$$\text{size}(E) = [d_1 \ d_2 \ \dots \ d_D \ n]$$

- The $(:)$ operator reshapes a matrix to a 1D vector(the same order as storage in memory).

- Sorting arrays:
 - $[value, index] = sort(V, 'ascend')$
 - $V_{asc} = V(index)$; re orders(warps) V in the order of index (ascending) values ($value = V_{asc}$)
- Finding indicies
 - say $P = [V == max(V)]$ P is a vector of 0's and 1's.
 - $index = find(P)$: finds the nonzero elements of the P vector.
 - The same for matrices:
 $index = find(A(:) == max(A(:)))$

- A and B are square and equally sized 100 x 100.
- Similar to $C = AB$: $C_{i,j} = \sum_k A_{i,k} B_{k,j}$
- write one line of code to do:
 - $W_{i,j} = \sum_k A_{k,i} B_{j,k}$
 - $W_{i,j} = \sum_k A_{i,k} B_{j,k}$
 - $W_{i,j} = \sum_k A_{i,j} B_{j,k}$
 - $W_{i,j} = \sum_k A_{101-i,k} B_{101-k,j}$

- A and B are square and equally sized 100×100 .
- Similar to $C = AB$: $C_{i,j} = \sum_k A_{i,k} B_{k,j}$
- write one line of code to do:
 - $W_{i,j} = \sum_k A_{k,i} B_{j,k} = (B * A)'$;
 - $W_{i,j} = \sum_k A_{i,k} B_{j,k}$
 - $W_{i,j} = \sum_k A_{i,j} B_{j,k}$
 - $W_{i,j} = \sum_k A_{101-i,k} B_{101-k,j}$

- A and B are square and equally sized 100×100 .
- Similar to $C = AB$: $C_{i,j} = \sum_k A_{i,k} B_{k,j}$
- write one line of code to do:
 - $W_{i,j} = \sum_k A_{k,i} B_{j,k} = (B * A)'$;
 - $W_{i,j} = \sum_k A_{i,k} B_{j,k} = (B * A')'$;
 - $W_{i,j} = \sum_k A_{i,j} B_{j,k}$
 - $W_{i,j} = \sum_k A_{101-i,k} B_{101-k,j}$

- A and B are square and equally sized 100 x 100.
- Similar to $C = AB$: $C_{i,j} = \sum_k A_{i,k} B_{k,j}$
- write one line of code to do:
 - $W_{i,j} = \sum_k A_{k,i} B_{j,k} = (B * A)'$;
 - $W_{i,j} = \sum_k A_{i,k} B_{j,k} = (B * A')'$;
 - $W_{i,j} = \sum_k A_{i,j} B_{j,k} = A. * (\text{repmat}(\text{sum}(B, 2)', [100, 1]));$
 - $W_{i,j} = \sum_k A_{101-i,k} B_{101-k,j}$

- Mahalanobis Distance (many to one)

$$D_M(x, y, \sigma) = (x - y)^T \sigma^{-1} (x - y)$$

- Finding the closest points, 2 sets

$$C_2(X, Y) = \operatorname{argmin}_{x \in X, y \in Y} \|x - y\|^2$$

- Finding the closest (smallest triangle) points, 3 sets

$$C_3(X, Y, Z) = \operatorname{argmin}_{x \in X, y \in Y, z \in Z} \left(\|x - y\| \|z - y\| - \frac{z - y}{\|z - y\|} \cdot (x - y) \right)^2$$