QR decomposition using GPU and OpenCL

In this example, we will create a random 4x5 matrix using uniform distribution and calculate its QR decomposition matrix. The calculation will be achieved using the GPU card and OpenCL with a group of MatDeck functions that incorporate ArrayFire functionalities.

First, we will set the environment to use the GPU for calculations. Using the function, afp_supported_backends, a list of all supported backends that can be used for calculations will be produced. In our case, calculations can be made on the CPU, using OpenCL or CUDA framework.

$$\text{afp\_supported\_backends}(\,) = \begin{bmatrix} \text{"cpu"} \\ \text{"opencl"} \\ \text{"cuda"} \end{bmatrix}$$

Default environment for calculations is the CPU. We can change the current environment with the function, afp_set_backend, and check which environment is currently in use with the afp_backend function.

$$\text{afp\_set\_backend}(\text{"opencl"}) = \text{true}$$

$$\text{afp\_backend}(\,) = \text{"opencl"}$$

In each environment, there can be several devices which support calculations within it. To check the number of devices which support calculations in the current environment, use the function, afp_get_device_count, and the functions afp_get_device and afp_set_device to check/change current device.

$$\text{afp\_get\_device\_count}(\,) = 3$$

$$\text{afp\_get\_device}(\,) = 1$$

$$\text{afp\_set\_device}(1) = \text{true}$$

To display information about currently selected devices, use the function afp_device_info

$$\text{afp\_device\_info}(\,) = \begin{bmatrix} \text{"Intel(R)\_HD\_Graphics\_620"} \\ \text{"OpenCL"} \\ \text{"Intel(R) OpenCL"} \\ \text{"2.1"} \end{bmatrix}$$

Finally, we have set the OpenCL as a calculation backend and set the device with number 1 - Intel CPU with OpenCL support as a device on which we will do all calculations.

Let's create a uniformly random 4x5 matrix with real values.

$$A := \text{afp\_randu}(4\ ,\ 5\ ,\ \text{"real"})$$

We can print the variable A to check that the input matrix is generated.

$$A = \begin{bmatrix} 0.995 & 0.347 & 0.192 & 0.853 & 0.613 \\ 0.615 & 0.005 & 0.647 & 0.225 & 0.902 \\ 0.829 & 0.227 & 0.285 & 0.552 & 0.745 \\ 0.87 & 0.386 & 0.4 & 0.082 & 0.37 \end{bmatrix}$$

Now, we can do the QR decomposition calculations on matrix A and place the resulting vector in variable B. Resulting vector contains the orthogonal matrix Q and upper triangle matrix R.

$$B := \text{afp\_qr}(A)$$

$$B = \begin{bmatrix} \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0.619 & 1 & 0 & 0 \\ 0.874 & -0.396 & 1 & 0 \\ 0.833 & 0.294 & -0.068 & 1 \end{bmatrix} & \begin{bmatrix} 0.995 & 0.347 & 0.192 & 0.853 & 0.613 \\ 0 & -0.21 & 0.528 & -0.303 & 0.523 \\ 0 & 0 & 0.441 & -0.784 & 0.041 \\ 0 & 0 & 0 & -0.123 & 0.084 \end{bmatrix} \end{bmatrix}$$

There are separate functions for every member of the resulting vector.
Function afp_qr_q will calculate the orthogonal matrix, Q

$$\text{afp\_qr\_q}(A) = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0.619 & 1 & 0 & 0 \\ 0.874 & -0.396 & 1 & 0 \\ 0.833 & 0.294 & -0.068 & 1 \end{bmatrix}$$

Function afp_qr_r will calculate the upper triangular matrix, R

$$\text{afp\_qr\_r}(A) = \begin{bmatrix} 0.995 & 0.347 & 0.192 & 0.853 & 0.613 \\ 0 & -0.21 & 0.528 & -0.303 & 0.523 \\ 0 & 0 & 0.441 & -0.784 & 0.041 \\ 0 & 0 & 0 & -0.123 & 0.084 \end{bmatrix}$$