

# QR decomposition using GPU and CUDA

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 Nvidia GPU card and CUDA 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.

```
afp_supported_backends() = [ "cpu"  
                           "opencl"  
                           "cuda" ]
```

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.

```
afp_set_backend("cuda") = true  
afp_backend() = "cuda"
```

In each environment, there can be several devices which support the 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.

```
afp_get_device_count() = 1  
  
afp_get_device() = 0  
afp_set_device(0) = true
```

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

```
afp_device_info() = [ "NVIDIA_GeForce_940MX"  
                     "CUDA"  
                     "v11.2"  
                     "5.0" ]
```

Finally, we have set CUDA as a calculation backend and set the device with number 0 - Nvidia GeForce GPU card with CUDA support as a device on which we will do all calculations.

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

```
A := afp_randu(4, 5, "real")
```

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

$$A = \begin{bmatrix} 0.785 & 0.842 & 0.702 & 0.29 & 0.995 \\ 0.987 & 0.722 & 0.747 & 0.523 & 0.615 \\ 0.113 & 0.328 & 0.339 & 0.997 & 0.829 \\ 0.454 & 0.964 & 0.688 & 0.753 & 0.87 \end{bmatrix}$$

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 the upper triangle matrix, R.

`B := afp_qr(A)`

$$B = \begin{bmatrix} \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0.460 & 1 & 0 & 0 \\ 0.115 & 0.389 & 1 & 0 \\ 0.796 & 0.422 & -0.316 & 1 \end{bmatrix} & \begin{bmatrix} 0.987 & 0.722 & 0.747 & 0.523 & 0.615 \\ 0 & 0.632 & 0.345 & 0.512 & 0.586 \\ 0 & 0 & 0.119 & 0.738 & 0.530 \\ 0 & 0 & 0 & -0.109 & 0.425 \end{bmatrix} \end{bmatrix}$$

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

$$\text{afp\_qr\_q}(A) = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0.460 & 1 & 0 & 0 \\ 0.115 & 0.389 & 1 & 0 \\ 0.796 & 0.422 & -0.316 & 1 \end{bmatrix}$$

Function `afp_qr_r` will calculate the upper triangular matrix R

$$\text{afp\_qr\_r}(A) = \begin{bmatrix} 0.987 & 0.722 & 0.747 & 0.523 & 0.615 \\ 0 & 0.632 & 0.345 & 0.512 & 0.586 \\ 0 & 0 & 0.119 & 0.738 & 0.530 \\ 0 & 0 & 0 & -0.109 & 0.425 \end{bmatrix}$$