

# 1 Testing Classical Gram-Schmidt

## 1.1 Figures for Q Matrix in Assembly

| Address    | +0 | +1 | +2 | +3 |            |    |    |    |
|------------|----|----|----|----|------------|----|----|----|
| 0x100000A8 | 69 | 41 | B2 | BE |            |    |    |    |
| 0x100000A4 | 3A | CD | 13 | 3F |            |    |    |    |
| 0x100000A0 | 05 | FB | 96 | BE |            |    |    |    |
| 0x1000009C | C1 | D9 | 4C | 3F |            |    |    |    |
| 0x10000098 | 0F | B1 | 05 | 3F |            |    |    |    |
| 0x10000094 | 00 | 00 | 00 | 00 |            |    |    |    |
| 0x10000090 | 08 | FB | 16 | 3E |            |    |    |    |
| 0x1000008C | BD | D9 | CC | BE |            |    |    |    |
| 0x10000088 | 6A | 41 | 32 | 3F |            |    |    |    |
| 0x10000084 | 3A | CD | 13 | 3F |            |    |    |    |
|            |    |    |    |    | 0x100000C0 | C8 | B9 | 3C |
|            |    |    |    |    | 0x100000BC | D4 | 16 | 36 |
|            |    |    |    |    | 0x100000B8 | 69 | 41 | B2 |
|            |    |    |    |    | 0x100000B4 | 3A | CD | 13 |
|            |    |    |    |    | 0x100000B0 | 07 | FB | 16 |
|            |    |    |    |    | 0x100000AC | 9E | 9C | E3 |
|            |    |    |    |    | 0x100000A8 | 69 | 41 | B2 |
|            |    |    |    |    |            |    |    | BE |

Figure 1: Q Matrix in Assembly - Part 1 and Part 2

## 1.2 Figures for R Matrix in Assembly

| Address    | +0 | +1 | +2 | +3 |            |    |    |    |
|------------|----|----|----|----|------------|----|----|----|
| 0x100000F0 | 06 | 54 | 9F | BE |            |    |    |    |
| 0x100000EC | 74 | E0 | 82 | 3F |            |    |    |    |
| 0x100000E8 | 00 | 00 | 00 | 00 |            |    |    |    |
| 0x100000E4 | 00 | 00 | 00 | 00 |            |    |    |    |
| 0x100000E0 | 31 | 67 | 16 | 40 |            |    |    |    |
| 0x100000DC | 53 | 1D | A7 | 3F |            |    |    |    |
| 0x100000D8 | F1 | 19 | F5 | 3F |            |    |    |    |
| 0x100000D4 | 00 | 00 | 00 | 00 |            |    |    |    |
| 0x100000D0 | D7 | B3 | 5D | 40 |            |    |    |    |
| 0x100000CC | 66 | 90 | 8A | 40 |            |    |    |    |
| 0x100000C8 | 3A | CD | 93 | 3F |            |    |    |    |
| 0x100000C4 | D7 | B3 | DD | 3F |            |    |    |    |
|            |    |    |    |    | 0x10000100 | 28 | 99 | CF |
|            |    |    |    |    | 0x100000FC | 00 | 00 | 3F |
|            |    |    |    |    | 0x100000F8 | 00 | 00 | 00 |
|            |    |    |    |    | 0x100000F4 | 00 | 00 | 00 |
|            |    |    |    |    | 0x100000F0 | 06 | 54 | 9F |
|            |    |    |    |    |            |    |    | BE |

Figure 2: R Matrix in Assembly - Part 1 and Part 2

### 1.3 C++ Code Results

```
Classical Gram-Schmidt Q Matrix:  
  
[0.57735, 0.696311, -0.400099, 0.147442]  
[0, 0.522233, 0.800198, -0.294884]  
[0.57735, -0.348155, 0.444554, 0.589768]  
[0.57735, -0.348155, -0.0444554, -0.73721]  
  
Classical Gram-Schmidt R Matrix:  
  
[1.73205, 1.1547, 4.33013, 3.4641]  
[0, 1.91485, 1.30558, 2.35005]  
[0, 0, 1.02247, -0.311188]  
[0, 0, 0, 1.62186]
```

Figure 3: C++ Code Results

### 1.4 Q and R Matrices from C++ Code

$$Q = \begin{bmatrix} 0.57735 & 0.696311 & -0.400099 & 0.147442 \\ 0 & 0.522233 & 0.800198 & -0.294884 \\ 0.57735 & -0.348155 & 0.444554 & 0.589768 \\ 0.57735 & -0.348155 & -0.0444554 & -0.73721 \end{bmatrix}$$

$$R = \begin{bmatrix} 1.73205 & 1.1547 & 4.33013 & 3.4641 \\ 0 & 1.91485 & 1.30558 & 2.35005 \\ 0 & 0 & 1.02247 & -0.311188 \\ 0 & 0 & 0 & 1.62186 \end{bmatrix}$$

## 2 Testing Modified Gram-Schmidt

### 2.1 Figures for Q Matrix in Assembly

| Address    | +0 | +1 | +2 | +3 |            | C7 | B9 | 3C | BF |
|------------|----|----|----|----|------------|----|----|----|----|
| 0x100000A8 | 69 | 41 | B2 | BE | 0x100000C0 |    |    |    |    |
| 0x100000A4 | 3A | CD | 13 | 3F | 0x100000BC | BD | 16 | 36 | BD |
| 0x100000A0 | 06 | FB | 96 | BE | 0x100000B8 | 69 | 41 | B2 | BE |
| 0x1000009C | C0 | D9 | 4C | 3F | 0x100000B4 | 3A | CD | 13 | 3F |
| 0x10000098 | 0F | B1 | 05 | 3F | 0x100000B0 | 06 | FB | 16 | 3F |
| 0x10000094 | 00 | 00 | 00 | 00 | 0x100000AC | A1 | 9C | E3 | 3E |
| 0x10000090 | 08 | FB | 16 | 3E |            |    |    |    |    |
| 0x1000008C | BA | D9 | CC | BE |            |    |    |    |    |
| 0x10000088 | 6A | 41 | 32 | 3F |            |    |    |    |    |
| 0x10000084 | 3A | CD | 13 | 3F |            |    |    |    |    |

Figure 4: Q Matrix in Assembly - Part 1 and Part 2

### 2.2 Figures for R Matrix in Assembly

| Address    | +0 | +1 | +2 | +3 |            | 0x10000100 | 28 | 99 | CF | 3F |
|------------|----|----|----|----|------------|------------|----|----|----|----|
| 0x100000F0 | 04 | 54 | 9F | BE | 0x100000FC | 00         | 00 | 00 | 00 |    |
| 0x100000EC | 74 | E0 | 82 | 3F | 0x100000F8 | 00         | 00 | 00 | 00 |    |
| 0x100000E8 | 00 | 00 | 00 | 00 | 0x100000F4 | 00         | 00 | 00 | 00 |    |
| 0x100000E4 | 00 | 00 | 00 | 00 | 0x100000F0 | 04         | 54 | 9F | BE |    |
| 0x100000E0 | 31 | 67 | 16 | 40 |            |            |    |    |    |    |
| 0x100000DC | 53 | 1D | A7 | 3F |            |            |    |    |    |    |
| 0x100000D8 | F1 | 19 | F5 | 3F |            |            |    |    |    |    |
| 0x100000D4 | 00 | 00 | 00 | 00 |            |            |    |    |    |    |
| 0x100000D0 | D7 | B3 | 5D | 40 |            |            |    |    |    |    |
| 0x100000CC | 66 | 90 | 8A | 40 |            |            |    |    |    |    |
| 0x100000C8 | 3A | CD | 93 | 3F |            |            |    |    |    |    |
| 0x100000C4 | D7 | B3 | DD | 3F |            |            |    |    |    |    |

Figure 5: R Matrix in Assembly - Part 1 and Part 2

### 2.3 C++ Code Results

```
Modified Gram-Schmidt Q Matrix:  
  
[0.57735, 0.696311, -0.400099, 0.147442]  
[0, 0.522233, 0.800198, -0.294884]  
[0.57735, -0.348155, 0.444554, 0.589768]  
[0.57735, -0.348155, -0.0444554, -0.73721]  
  
Modified Gram-Schmidt R Matrix:  
  
[1.73205, 1.1547, 4.33013, 3.4641]  
[0, 1.91485, 1.30558, 2.35005]  
[0, 0, 1.02247, -0.311188]  
[0, 0, 0, 1.62186]
```

Figure 6: C++ Code Results

### 2.4 Q and R Matrices from C++ Code

$$Q = \begin{bmatrix} 0.57735 & 0.696311 & -0.400099 & 0.147442 \\ 0 & 0.522233 & 0.800198 & -0.294884 \\ 0.57735 & -0.348155 & 0.444554 & 0.589768 \\ 0.57735 & -0.348155 & -0.0444554 & -0.73721 \end{bmatrix}$$

$$R = \begin{bmatrix} 1.73205 & 1.1547 & 4.33013 & 3.4641 \\ 0 & 1.91485 & 1.30558 & 2.35005 \\ 0 & 0 & 1.02247 & -0.311188 \\ 0 & 0 & 0 & 1.62186 \end{bmatrix}$$

### 3 Generalized Using Logbase2

```
C:\> Users > AUB > Desktop > 321_project > --- generalized_logbase2.s
380 addi x16,x0,0
381 lop3:
382 bge x16,x13,end_lop3 ↗
383 sll x19,x16,x31 ## 1*4
384 add x19,x19,x15 ## 1*4+k
385 slli x19,x19,2 ## [1][k]
386 add x20,x19,x11 ## x20=8Q[i][k]
387 f1w f1,0(x20) ## f1=Q[i][k]
388 fdiv.s f1,f1,f0 ## Q[i][k]/R[k][k]
389 fsr f1,0(x20) ## updated Q[i][k]
390 addi x16,x16,1
391 j lop3
392 end_lop3:
393 addi x30,x15,1 ## j=k+1
394 lop4:
395 bge x30,x13,end_lop4 ↗
396 sll x20,x15,x31 ## k*4
397 add x20,x20,x30 ## k*4+j
398 slli x20,x20,2 ## [k][j]
399 add x20,x20,x14 ## &R[k][j]
400 la x7,zero_float
401 f1w f5,0(x7)
402 fsr f5,0(x20) ## R[k][j]=0
403 addi x16,x0,0 ## i=0
404 lop5: ## x20=RR[k][j] ↗
405 bge x16,x13,end_lop5
406 sll x19,x16,x31 ## x19=1*4
407 add x22,x19,x15 ## x20=i*4+k
408 slli x22,x22,2 ## [i][k]
409 add x21,x19,x30 ## x21=i*4+j
410 slli x21,x21,2 ## [i][j]
411 add x22,x22,x11 ## RQ[i][k]
412 add x21,x21,x11 ## RQ[i][j]
413 f1w f0,0(x21) ## f0=Q[i][j]
414 f1w f1,0(x22) ## f1=Q[i][k]
415 f1w f2,0(x20) ## f2=R[k][j]
416 fmul.s f0,f0,f1,f1 ## f1=f1*f2+f0*f1
417
```

Figure 7: Generalized Logbase2 - Figure 1

```
logbase2: ## final value in x11, input in x14
addi sp,sp,-12
sw x10,0(sp)
sw x6,4(sp)
sw x1,8(sp)
li x10,1 ## initial , maybe its 2 so logbase2(2)=1
li x6,2 ## start with 2

d1:
beq x6,x14,doned1
slli x6,x6,1
addi x10,x10,1
j d1
doned1:
addi x11,x10,0
lw x10,0(sp)
lw x6,4(sp)
lw x1,8(sp)
jalr x0,0(x1)
```

Figure 8: Generalized Logbase2 - Figure 2

## 4 Unrolling

### 4.1 Unrolling One Loop

```
loop5: ## x20=&R[k][j]
    bge x16,x13,end_lop5
    slli x19,x16,2 ## x19=i*4
    add x22,x19,x15 ## x20=i*4+k
    slli x22,x22,2 ## [i][k]
    add x21,x19,x30 ## x21=i*4+j
    slli x21,x21,2 ## [i][j]
    add x22,x22,x11 ## &Q[i][k]
    add x21,x21,x11 ## &Q[i][j]
    flw f0,0(x21) ## f0=Q[i][j]
    flw f1,0(x22) ## f1=Q[i][k]
    flw f2,0(x20) ## f2=R[k][j]
    fmul.s f0,f0,f1 ## Q[i][k]*Q[i][j]
    fadd.s f2,f0,f2 ## R[k][j]+Q[i][k]*Q[i][j]
    fsw f2,0(x20) ## R[k][j] += Q[i][k] * Q[i][j]
    addi x16,x16,1
    j loop5
end_lop5:
```

Figure 9: Before Unrolling Loop

```

## First iteration that can be done in independent core
li x19,0 ## [0]
add x22,x19,x15 ## 0+k
slli x22,x22,2 ## [0][k] single precision
add x21,x19,x30 ## 0+j
slli x21,x21,2 ## [0][j]
add x22,x22,x11 ## &Q[0][k]
add x21,x21,x11 ## &Q[0][j]
flw f0,0(x22) ## f0=Q[0][k]
flw f1,0(x21) ## f1=Q[0][j]
flw f2,0(x20) ## f2=R[k][j]
fmul.s f0,f0,f1 ## Q[0][k]*Q[0][j]
fadd.s f2,f0,f2 ## R[k][j]+Q[0][k]*Q[0][j]
fsw f2,0(x20) ## R[k][j] += Q[0][k] * Q[0][j]
##

##Second iteration that can be done in independent core
li x19,4 ## [1]
add x22,x19,x15 ## 4+k
slli x22,x22,2 ## [1][k] single precision
add x21,x19,x30 ## 1+j
slli x21,x21,2 ## [1][j]
add x22,x22,x11 ## &Q[1][k]
add x21,x21,x11 ## &Q[1][j]
flw f0,0(x22) ## f0=Q[1][k]
flw f1,0(x21) ## f1=Q[1][j]
flw f2,0(x20) ## f2=R[k][j]
fmul.s f0,f0,f1 ## Q[1][k]*Q[1][j]
fadd.s f2,f0,f2 ## R[k][j]+Q[1][k]*Q[1][j]
fsw f2,0(x20) ## R[k][j] += Q[1][k] * Q[1][j]
##

```

Figure 10: After Unrolling Loop

## 4.2 Unrolling Multiple Loops

```
9  lopp6: ## x20=& R[k][j]
0  bge x16,x13,end_lopp6
1  slli x22,x16,2 ## i*4
2  add x22,x22,x30 ## i*4+j
3  slli x22,x22,2 ## [i][j]
4  slli x21,x16,2 ## i*4
5  add x21,x21,x15 ## i*4+k
6  slli x21,x21,2 ## [i][k]
7  add x22,x22,x11 ## &Q[i][j]
8  add x21,x21,x11 ## &Q[i][k]
9  flw f0,0(x21) ## Q[i][k]
0  flw f1,0(x22)## Q[i][j]
1  flw f2,0(x20) ## R[k][j]
2  fmul.s f2,f0,f2 ## R[k][j] * Q[i][k]
3  fsub.s f1,f1,f2 ## Q[i][j] - R[k][j] * Q[i][k]
4  fsw f1,0(x22) ## Q[i][j] -= R[k][j] * Q[i][k]
5  addi x16,x16,1
6  j lopp6
7  .L1.6
```

Figure 11: Before Unrolling second Loop - Figure 3

```

## First iteration that could be done on independent cores
##, once R[k][j] is finalized above
li x22,0 ## i=0
add x22,x22,x30 ## i*0+j
slli x22,x22,2 ## [0][j]
li x21,0
add x21,x21,x15 ## i*0+k
slli x21,x21,2 ## [0][k]
add x22,x22,x11 ## &Q[0][j]
add x21,x21,x11 ## & Q[0][k]
f1w f0,0(x21) ## Q[0][k]
f1w f1,0(x22)## Q[0][j]
f1w f2,0(x20) ## R[k][j]
fmul.s f2,f0,f2 ## R[k][j] * Q[0][k]
fsub.s f1,f1,f2 ## Q[0][j] - R[k][j] * Q[0][k]
fsw f1,0(x22) ## Q[0][j] -= R[k][j] * Q[0][k]
##

## Second iteration that could be done on independent cores
##, once R[k][j] is finalized above
li x22,4 ## i=1
add x22,x22,x30 ## 4+j
slli x22,x22,2 ## [1][j]
li x21,4
add x21,x21,x15 ## 4+k
slli x21,x21,2 ## [1][k]
add x22,x22,x11 ## 8Q[1][j]
add x21,x21,x11 ## & Q[1][k]
f1w f0,0(x21) ## Q[1][k]
f1w f1,0(x22)## Q[1][j]
f1w f2,0(x20) ## R[k][j]
fmul.s f2,f0,f2 ## R[k][j] * Q[1][k]
fsub.s f1,f1,f2 ## Q[1][j] - R[k][j] * Q[1][k]
fsw f1,0(x22) ## Q[1][j] -= R[k][j] * Q[1][k]
##

```

Figure 12: After Unrolling second Loop - Figure 4

## 5 QR Extension

```

1225    ## XEE: q_fj > m m from memory
1226    solveforx: ## given y is in x8, H is in x24
1227    ## QR DECOMPOSE H
1228    ## where modified takes A in x10,
1229    ## and result it in x11
1230    ## and x14
1231    addi x10,x24,0 ## A--> H
1232    call modified_granny ## now x11--> Q, x14-> R
1233    addi x10,x23,0 ## result in Q_t
1234    addi x26,x11,0 ## input Q in x26
1235    call transposeMatrix ## our result in x10
1236    ## auto auto tghyrt ma3a x23 , same address :)
1237    ## our y is in x8, our Q_t is in x10
● 1238    addi x11,x25,0 ## now x11 holds new address for y_prime
1239    addi x24,x8,0 ## now x24--> holds y
1240    addi x27,x10,0
1241    call multiplyMatrixVector
1242    ## y_prime is in x11, R--> x14
● 1243    addi x18,x11,0 ## y_prime --> x18
1244    addi x10,x28,0
1245    call backwardSubstitution ## our result in x10
● 1246    li a0,10
1247    ecall
1248

```

Figure 13: QR Extension - Assembly Code for Function

| Address    | +0 | +1 | +2 | +3 |            |    |    |    |
|------------|----|----|----|----|------------|----|----|----|
| 0x10000180 | BD | 16 | 36 | BD |            |    |    |    |
| 0x1000017C | A1 | 9C | E3 | 3E |            |    |    |    |
| 0x10000178 | C0 | D9 | 4C | 3F |            |    |    |    |
| 0x10000174 | 8A | D9 | CC | BE |            |    |    |    |
| 0x10000170 | 69 | 41 | B2 | BE |            |    |    |    |
| 0x1000016C | 69 | 41 | B2 | BE |            |    |    |    |
| 0x10000168 | 0F | B1 | 05 | 3F |            |    |    |    |
| 0x10000164 | 6A | 41 | 32 | 3F |            |    |    |    |
| 0x10000160 | 3A | CD | 13 | 3F |            |    |    |    |
| 0x1000015C | 3A | CD | 13 | 3F |            |    |    |    |
| 0x10000158 | 00 | 00 | 00 | 00 |            |    |    |    |
| 0x10000154 | 3A | CD | 13 | 3F |            |    |    |    |
|            |    |    |    |    | 0x10000190 | C7 | B9 | 3C |
|            |    |    |    |    | 0x1000018C | 06 | FB | 16 |
|            |    |    |    |    | 0x10000188 | 06 | FB | 96 |
|            |    |    |    |    | 0x10000184 | 0B | FB | 16 |
|            |    |    |    |    | 0x10000180 | BD | 16 | 3E |
|            |    |    |    |    |            |    |    | BD |

Figure 14: QR Extension - Transposing Q Matrix

|            |    |    |    |    |
|------------|----|----|----|----|
| 0x10000140 | A2 | 1B | 84 | BF |
| 0x1000013C | 8E | 3E | 33 | 40 |
| 0x10000138 | C6 | D1 | 5F | 3F |
| 0x10000134 | 30 | 3A | CB | 40 |

Figure 15: QR Extension - Result of MultiplyMatrix Vector (y')

| <b>Address</b> | <b>+0</b> | <b>+1</b> | <b>+2</b> | <b>+3</b> |
|----------------|-----------|-----------|-----------|-----------|
| 0x10000150     | B6        | E8        | 22        | BF        |
| 0x1000014C     | C0        | E8        | 22        | 40        |
| 0x10000148     | 13        | 00        | 00        | BF        |
| 0x10000144     | 00        | A3        | 8B        | BF        |

Figure 16: QR Extension - Result of Backward Substitution (x)

```

Transposed Q Matrix:
[0.57735, 0, 0.57735, 0.57735]
[0.696311, 0.522233, -0.348155, -0.348155]
[-0.400099, 0.800198, 0.444554, -0.0444554]
[0.147442, -0.294884, 0.589768, -0.73721]

Result of Q_T * y (y_prime):
[6.35085, 0.870388, 2.80069, -1.03209]

Result of backward substitution (x):
[-1.09091, -0.5, 2.54545, -0.636364]

```

Figure 17: QR Extension - C++ Code Results

## 5.1 Transposed Q Matrix from C++ Code

$$Q^T = \begin{bmatrix} 0.57735 & 0 & -0.400099 & 0.147442 \\ 0.57735 & 0.522233 & 0.800198 & -0.294884 \\ 0.57735 & -0.348155 & 0.444554 & 0.589768 \\ 0.57735 & -0.348155 & -0.0444554 & -0.73721 \end{bmatrix}$$

## 5.2 y' Vector from C++ Code

$$y' = \begin{bmatrix} 6.35085 \\ 0.870388 \\ 2.80069 \\ -1.03209 \end{bmatrix}$$

## 5.3 x Vector from C++ Code

$$x = \begin{bmatrix} -1.09091 \\ -0.5 \\ 2.54545 \\ -0.636364 \end{bmatrix}$$

## 6 Caching

```
17    loop2:  
18        bge x16, x13, end_loop2  
19        slli x19, x16, 2  
20        add x19, x17, x19 # &q[i]  
21        slli x20, x16, 2 #  
22        add x20, x20, x15  
23        slli x20, x20, 2  
24        add x20, x20, x10 # &A[i][k]  
25        flw f1, 0(x20)  
26        fsw f1, 0(x19) # q[i] = A[i][k]  
27        addi x16, x16, 1  
28        j loop2  
29    end_loop2:
```

Figure 18: Code Before Caching

```
390  
391    lopcache:  
392        bge x16, x13, end_lopcache  
393        slli x19, x16, 2  
394        add x19, x4, x19 # &a_cache[i]  
395        slli x20, x16, 2 #  
396        add x20, x20, x15  
397        slli x20, x20, 2  
398        add x20, x20, x11 # &Q[i][k]  
399        flw f1, 0(x20)  
400        fsw f1, 0(x19) # a_cache[i] = A[i][k]  
401        addi x16, x16, 1  
402        j lopcache  
403    end_lopcache:  
404    li x16, 0  
405    lop2:  
406        bge x16, x13, end_lop2  
407        slli x19, x16, 2  
408        add x19, x17, x19 # &q[i]  
409        slli x20, x16, 2 #  
410        add x20, x20, x4  
411        flw f1, 0(x20)  
412        fsw f1, 0(x19) # q[i] = a_cache[i]  
413        addi x16, x16, 1  
414        j lop2  
415    end_lop2:  
416  
417
```

Figure 19: Code After Caching

## 7 Verifications

### 7.1 First Identity: $Q \times R = A$

|            | 24 | F1 | 78 | B1 |
|------------|----|----|----|----|
| 0x10000110 | 24 | F1 | 78 | B1 |
| 0x1000010C | FF | FF | FF | 3F |
| 0x10000108 | 6F | 0D | 14 | 33 |
| 0x10000104 | FF | FF | 7F | 3F |
| 0x10000100 | 00 | 00 | 00 | 40 |
| 0x100000FC | 00 | 00 | 20 | 40 |
| 0x100000F8 | 6F | 0D | 14 | 33 |
| 0x100000F4 | FF | FF | 7F | 3F |
| 0x100000F0 | 00 | 00 | 00 | 3F |
| 0x100000EC | 01 | 00 | C0 | 3F |
| 0x100000E8 | FF | FF | 7F | 3F |
| 0x100000E4 | 00 | 00 | 00 | 00 |
| 0x100000E0 | 00 | 00 | 80 | 40 |
| 0x100000DC | 00 | 00 | 40 | 40 |
| 0x100000D8 | 00 | 00 | 00 | 40 |
| 0x100000D4 | FF | FF | 7F | 3F |

Q \* R Matrix:

```
[1, 2, 3, 4]
[0, 1, 1.5, 0.5]
[1, 0, 2.5, 2]
[1, 0, 2, 0]
```

Original A Matrix:

```
[1, 2, 3, 4]
[0, 1, 1.5, 0.5]
[1, 0, 2.5, 2]
[1, 0, 2, 0]
```

Figure 20: First Identity:  $Q \times R = A$  - Assembly (left) and C++ (right)

| RUN AND DEBUG                  | ... |
|--------------------------------|-----|
| Float                          | ... |
| f08: 0x00000000                |     |
| f10: 0x274ac312                |     |
| f11: 0x00000000                |     |
| f12: 0x00000000                |     |
| Mean Square Error: 2.81389e-15 |     |

Figure 21: Mean Squared Error for  $Q \times R = A$  - Assembly (left) and C++ (right)

## 7.2 Second Identity: $Q^T \times Q = I$

|            |    |    |    |    |
|------------|----|----|----|----|
| 0x10000150 | 01 | 00 | 80 | 3F |
| 0x1000014C | 87 | 90 | AB | 32 |
| 0x10000148 | 43 | 48 | 02 | 33 |
| 0x10000144 | 3A | CD | 13 | 33 |
| 0x10000140 | 87 | 90 | AB | 32 |
| 0x1000013C | 00 | 00 | 80 | 3F |
| 0x10000138 | 82 | E3 | 41 | 33 |
| 0x10000134 | 7E | 2D | F0 | 33 |
| 0x10000130 | 43 | 48 | 02 | 33 |
| 0x1000012C | 82 | E3 | 41 | 33 |
| 0x10000128 | 00 | 00 | 80 | 3F |
| 0x10000124 | 3A | CD | 13 | 33 |
| 0x10000120 | 3A | CD | 13 | 33 |
| 0x1000011C | 7E | 2D | F0 | 33 |
| 0x10000118 | 3A | CD | 13 | 33 |
| 0x10000114 | FF | FF | 7F | 3F |

|  |
|--|
| Q_T * Q Matrix:                                    |
| [1, -2.22045e-16, -4.96131e-16, -2.77556e-16]      |
| [-2.22045e-16, 1, -1.78677e-16, -5.55112e-16]      |
| [-4.96131e-16, -1.78677e-16, 1, 6.93889e-18]       |
| [-2.77556e-16, -5.55112e-16, 6.93889e-18, 1]       |
| Q_T * A Matrix:                                    |
| [1.73205, 1.1547, 4.33013, 3.4641]                 |
| [-3.33067e-16, 1.91485, 1.30558, 2.35005]          |
| [-8.88178e-16, -9.99201e-16, 1.02247, -0.311188]   |
| [-5.55112e-16, -1.44329e-15, -1.9984e-15, 1.62186] |

Figure 22: Second Identity:  $Q^T \times Q = I$  - Assembly (left) and C++ (right)

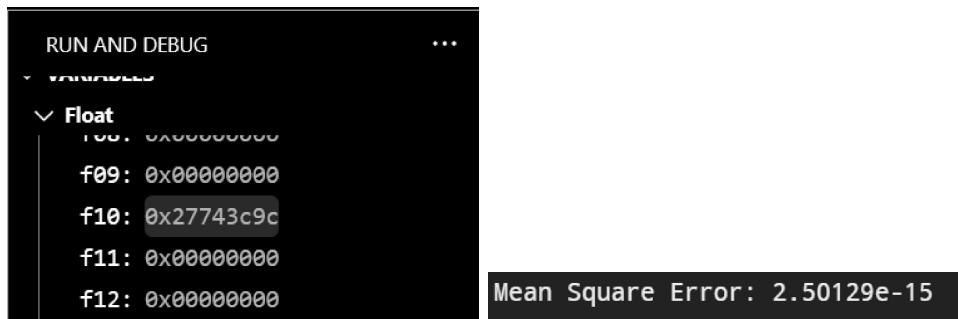


Figure 23: Mean Squared Error for  $Q^T \times Q = I$  - Assembly (left) and C++ (right)

### 7.3 Third Identity: $Q^T \times A = R$

|            |    |    |    |    |
|------------|----|----|----|----|
| 0x1000018C | 00 | 00 | 70 | 34 |
| 0x10000188 | 00 | 00 | C0 | 33 |
| 0x10000184 | 00 | 00 | 80 | 33 |
| 0x10000180 | F7 | 53 | 9F | BE |
| 0x1000017C | 78 | E0 | 82 | 3F |
| 0x10000178 | 00 | 00 | 80 | 34 |
| 0x10000174 | 00 | 00 | 50 | 34 |
| 0x10000170 | 32 | 67 | 16 | 40 |
| 0x1000016C | 54 | 1D | A7 | 3F |
| 0x10000168 | F2 | 19 | F5 | 3F |
| 0x10000164 | 00 | 00 | 80 | 33 |
| 0x10000160 | D7 | B3 | 5D | 40 |
| 0x1000015C | 66 | 90 | 8A | 40 |
| 0x10000158 | 3A | CD | 93 | 3F |
| 0x10000154 | D7 | B3 | DD | 3F |
| 0x10000150 | 01 | 00 | 80 | 3F |

|  |
|--|
| Q_T * A Matrix:                                    |
| [1.73205, 1.1547, 4.33013, 3.4641]                 |
| [-3.33067e-16, 1.91485, 1.30558, 2.35005]          |
| [-8.88178e-16, -9.99201e-16, 1.02247, -0.311188]   |
| [-5.55112e-16, -1.44329e-15, -1.9984e-15, 1.62186] |
| <br>   |
| [1.73205, 1.1547, 4.33013, 3.4641]                 |
| [0, 1.91485, 1.30558, 2.35005]                     |
| [0, 0, 1.02247, -0.311188]                         |
| [0, 0, 1.62186]                                    |

Figure 24: Third Identity:  $Q^T \times A = R$  - Assembly (left) and C++ (right)

| RUN AND DEBUG   | ...                            |
|-----------------|--------------------------------|
| ▼ Variables     | ...                            |
| ▼ Float         | ...                            |
| f09: 0x00000000 | Mean Square Error: 4.55469e-14 |
| f10: 0x294d2000 |                                |
| f11: 0x00000000 |                                |
| f12: 0x00000000 |                                |

Figure 25: Mean Squared Error for  $Q^T \times A = R$  - Assembly (left) and C++ (right)