# Wissenschaftliches Programmieren für Ingenieure

Übungsaufgabe mit Abgabe

**Name : YUNZE HAN**

**Immatrikulationsnummer: 2130541**

**Studiengang : Maschinenbau Master**

# Inhaltsverzeichnis

# Ausgabe

**Ausgabe Aufgabe 1**

Ausgabe Aufgabe 1: zu kopierender Abschnitt beginnt
QA=(1,2,3,4)

QB=QA+QA.conj() =(0,0,0,8)

QB=(QA-QA.conj())*2.=(4,8,12,0)

QC=(5,10,15,4)
QC=QA*QB= (16,32,48,-56)

QC rand = (7.82637e-06,0.131538,0.755605,0.45865)

Information in Funktion: Komponenten von Q
7.82637e-06 0.131538 0.755605 0.45865
---
Information in Funktion: Komponenten von Q
1 1 1 1
---
Ende des Ergebnisteils ==========

**Ausgabe Aufgabe 2**

**aufgabe2_output1**

Ortsvektor eingeben:Rotation: Achsenrichtung (wird normiert)
Winkel [Grad]
Quaternion X=(1,0,0,0)
QR=(0,1,0,6.12323e-17)
XP=(-1,0,1.22465e-16,0)

**aufgabe2_output2**

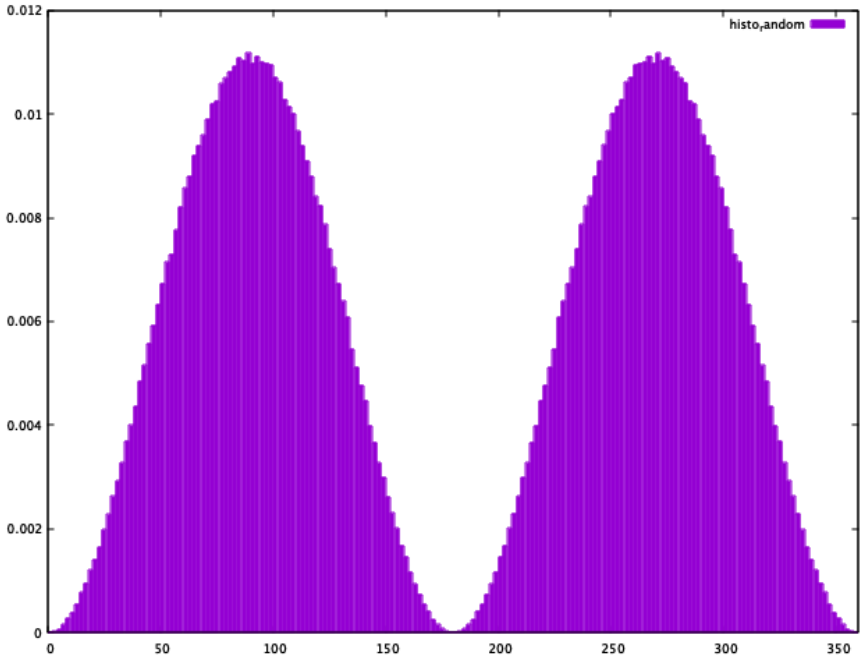
Ortsvektor eingeben:Rotation: Achsenrichtung (wird normiert)
Winkel [Grad]
Quaternion X=(1,0,0,0)
QR=(0.5,0.5,0.5,0.5)
XP=(1.11022e-16,-1.11022e-16,1,0)

**aufgabe2_output3**

Ortsvektor eingeben:Rotation: Achsenrichtung (wird normiert)
Winkel [Grad]
Quaternion X=(1,0,0,0)
QR=(0,0,0.707107,0.707107)
XP=(2.22045e-16,-1,0,0)

**Ausgabe Aufgabe 3**

**hist_random.dat**
(mit max =1500)
1 4.89215e-06
3 2.5795e-05
5 6.98243e-05
7 0.000168112
9 0.000277963
11 0.000397598
13 0.000551479
15 0.000784078
17 0.000958417
19 0.00121503
21 0.00140761
23 0.00165488
25 0.00199288
27 0.00229175
29 0.00263776
31 0.00292995
33 0.00327952
35 0.00369802
37 0.004
39 0.00436513
41 0.00485346
43 0.00515988
45 0.00558194
47 0.00592262
49 0.00632466
51 0.00672448
53 0.00714788
55 0.00729019
57 0.00776873
59 0.00820814
61 0.00857016
63 0.00879431
65 0.00920703
67 0.00938448
69 0.00959217
71 0.0099066
73 0.0101886
75 0.0102526
77 0.0105879
79 0.010692
81 0.0108094
83 0.0109131
85 0.0110829
87 0.011034



89 0.0111746
91 0.0109686
93 0.0110994
95 0.011006
97 0.0109669
99 0.0109433
101 0.0106991
103 0.0106137
105 0.0102762
107 0.0101343
109 0.0100044
111 0.00967178
113 0.0093956
115 0.00909451
117 0.00880543
119 0.00841005
121 0.00822682
123 0.00787414
125 0.00740227
127 0.00704158
129 0.00673071
131 0.00639226
133 0.00607872
135 0.0054672
137 0.00512653
139 0.00476629
141 0.00448076
143 0.00399333
145 0.00367578
147 0.00327507
149 0.00300511
151 0.00262886
153 0.00230954
155 0.00202179
157 0.00168868
159 0.00146409
161 0.0011759
163 0.000938848
165 0.000740938
167 0.000560374
169 0.000417167
171 0.000265066
173 0.000161886
175 8.7614e-05
177 2.62397e-05
179 4.44741e-06
181 4.44741e-06
183 2.62397e-05
185 8.7614e-05
187 0.000161886
189 0.000265066

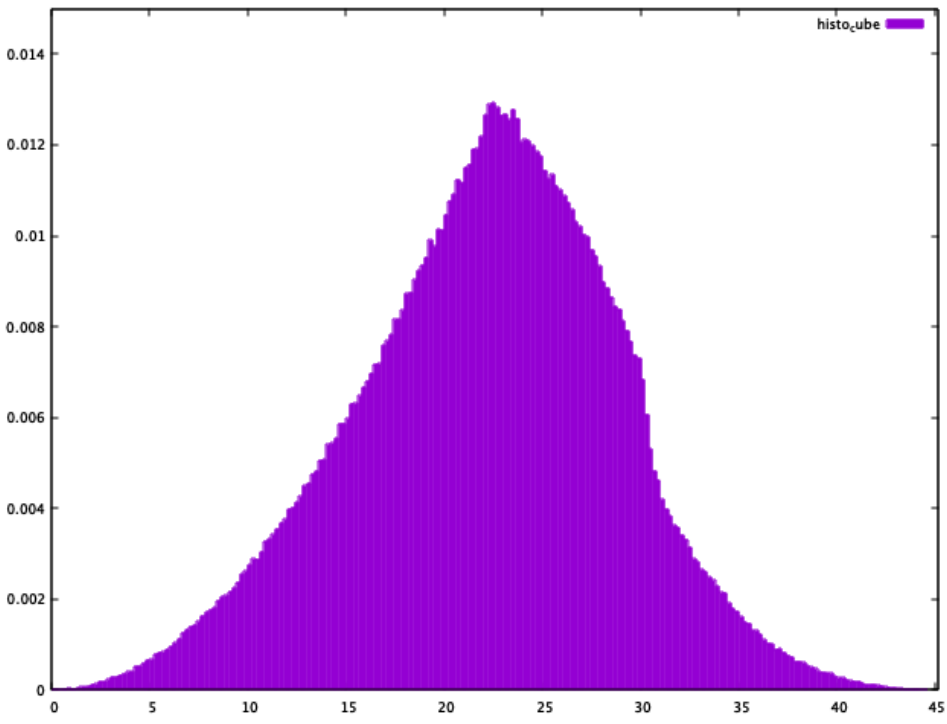
191 0.000417167
193 0.000559929
195 0.000741383
197 0.000938848
199 0.00117501
201 0.00146453
203 0.00168779
205 0.00202224
207 0.00230776
209 0.0026302
211 0.00300378
213 0.00327641
215 0.00367489
217 0.00399377
219 0.00447988
221 0.00476762
223 0.00512386
225 0.00546676
227 0.00607872
229 0.00639093
231 0.00673427
233 0.00704025
235 0.00740004
237 0.00787592
239 0.00822593
241 0.00841005
243 0.00880498
245 0.00909362
247 0.0093996
249 0.00966911
251 0.0100036
253 0.0101348
255 0.0102775
257 0.0106155
259 0.0106991
261 0.0109424
263 0.0109673
265 0.0110042
267 0.0111025
269 0.0109673
271 0.0111755
273 0.0110322
275 0.0110838
277 0.0109135
279 0.0108085
281 0.0106916
283 0.0105915
285 0.0102495
287 0.0101903
289 0.00990394
291 0.00959262

Name: YUNZE HAN Matrikelnummer:2130541

293 0.00938581
295 0.00920569
297 0.00879475
299 0.00856882
301 0.00820992
303 0.00777096
305 0.00728841
307 0.00714788
309 0.00672626
311 0.00632288
313 0.0059235
315 0.00558194
317 0.00515988
319 0.00485435
321 0.00436469
323 0.00399911
325 0.00369891
327 0.00327907
329 0.0029304
331 0.0026382
333 0.00229219
335 0.00199244
337 0.00165488
339 0.00140805
341 0.00121503
343 0.000957972
345 0.000784523
347 0.000551479
349 0.000397598
351 0.000277963
353 0.000168112
355 6.98243e-05
357 2.5795e-05
359 4.89215e-06

**hist_cube.dat**
(mit nmax=1500 , Breite= 0.2 ,Maxima = 0.0128)

0.1 8.89482e-07
0.3 8.89482e-07
0.5 4.44741e-06
0.7 1.42317e-05
0.9 3.73582e-05
1.1 3.11319e-05
1.3 3.82477e-05
1.5 7.3827e-05
1.7 7.64954e-05
1.9 8.71692e-05
2.1 0.000122748
2.3 0.000136091



2.5 0.000187681
2.7 0.000193907
2.9 0.000236602
3.1 0.000282855
3.3 0.00029264
3.5 0.000327329
3.7 0.000356682
3.9 0.000401156
4.1 0.000410051
4.3 0.000528352
4.5 0.000522126
4.7 0.000598621
4.9 0.000657327
5.1 0.000673338
5.3 0.000796086
5.5 0.000817434
5.7 0.000837892
5.9 0.00089304
6.1 0.000954414
6.3 0.00103891
6.5 0.00113231
6.7 0.00125595
6.9 0.00133244
7.1 0.00139293
7.3 0.00144007
7.5 0.00151924
7.7 0.00163131
7.9 0.00171937
8.1 0.00177274
8.3 0.00181099
8.5 0.00196398
8.7 0.00205915
8.9 0.00209117
9.1 0.00216144
9.3 0.00225039
9.5 0.00237581
9.7 0.00255281
9.9 0.00262219
10.1 0.00275206
10.3 0.00288815
10.5 0.00288459
10.7 0.0030447
10.9 0.00327952
11.1 0.003332
11.3 0.00342362
11.5 0.00354992
11.7 0.00368245
11.9 0.00376962
12.1 0.00399111
12.3 0.00401334
12.5 0.00413787

12.7 0.00426773
12.9 0.00450878
13.1 0.00455059
13.3 0.00474805
13.5 0.00482455
13.7 0.00504959
13.9 0.00507272
14.1 0.00540805
14.3 0.00544452
14.5 0.00554592
14.7 0.00585457
14.9 0.00585368
15.1 0.00598354
15.3 0.00630198
15.5 0.00631176
15.7 0.00647899
15.9 0.00665777
16.1 0.00679208
16.3 0.00697087
16.5 0.00716833
16.7 0.00719324
16.9 0.00758817
17.1 0.0076958
17.3 0.00782655
17.5 0.00816278
17.7 0.00816278
17.9 0.00836913
18.1 0.00872938
18.3 0.00874894
18.5 0.00903091
18.7 0.00923015
18.9 0.00935557
19.1 0.0095219
19.3 0.00990794
19.5 0.00977541
19.7 0.010141
19.9 0.0101196
20.1 0.0104648
20.3 0.0107663
20.5 0.0109175
20.7 0.0112368
20.9 0.0111674
21.1 0.0115046
21.3 0.0115588
21.5 0.0119004
21.7 0.0119182
21.9 0.0121903
22.1 0.0126582
22.3 0.0128993
22.5 0.0129313
22.7 0.0128237

22.9 0.0126493
23.1 0.0126724
23.3 0.012555
23.5 0.0127712
23.7 0.0125746
23.9 0.0120614
24.1 0.0121245
24.3 0.0120872
24.5 0.0119911
24.7 0.0118488
24.9 0.0117545
25.1 0.0114254
25.3 0.0112528
25.5 0.0113498
25.7 0.0110954
25.9 0.0110109
26.1 0.0108766
26.3 0.0107352
26.5 0.0105813
26.7 0.0103135
26.9 0.0102024
27.1 0.0100191
27.3 0.00996931
27.5 0.00969713
27.7 0.00955215
27.9 0.00933333
28.1 0.00897665
28.3 0.00885746
28.5 0.00865733
28.7 0.00844563
28.9 0.00838515
29.1 0.00812364
29.3 0.00790483
29.5 0.00767534
29.7 0.00737025
29.9 0.00730443
30.1 0.00683745
30.3 0.00607249
30.5 0.00530754
30.7 0.00481921
30.9 0.00461107
31.1 0.00420458
31.3 0.00398666
31.5 0.00382922
31.7 0.00363087
31.9 0.00358906
32.1 0.00341383
32.3 0.0033151
32.5 0.0031461
32.7 0.00288904
32.9 0.00282944

33.1 0.00265066
33.3 0.00260085
33.5 0.00249055
33.7 0.00242295
33.9 0.0022833
34.1 0.00215699
34.3 0.00214098
34.5 0.00192484
34.7 0.00179408
34.9 0.00173716
35.1 0.00160996
35.3 0.00149166
35.5 0.00145608
35.7 0.0013191
35.9 0.00131554
36.1 0.00122571
36.3 0.00110918
36.5 0.00103091
36.7 0.00101846
36.9 0.000900156
37.1 0.000917056
37.3 0.000834334
37.5 0.000757839
37.7 0.000723149
37.9 0.000621748
38.1 0.000622637
38.3 0.000603069
38.5 0.000547921
38.7 0.00051501
38.9 0.000479431
39.1 0.000402046
39.3 0.000386035
39.5 0.00037803
39.7 0.000375361
39.9 0.000305092
40.1 0.00027485
40.3 0.000273071
40.5 0.000220592
40.7 0.00020636
40.9 0.000193907
41.1 0.00018946
41.3 0.000163665
41.5 0.000128085
41.7 0.000128085
41.9 0.000112964
42.1 9.9622e-05
42.3 8.00534e-05
42.5 6.84901e-05
42.7 6.22637e-05
42.9 5.69268e-05
43.1 4.26951e-05

43.3 3.38003e-05
43.5 2.4016e-05
43.7 1.86791e-05
43.9 1.42317e-05
44.1 1.24527e-05
44.3 5.33689e-06
44.5 1.77896e-06
44.7 0
44.9 0

# Quellcode

### Aufgabe1

MyQuaternion.cpp

```
1.  #include "MyQuaternion.h"
2.  #include<iostream>
3.  #include <cstddef>
4.  #include <cassert>
5.  using namespace std;
6.
7.  //Konstruktor: MyQuaternion()
8.  MyQuaternion::MyQuaternion() {
9.
10.     this->dataPtr = new double[4];
11.
12. }
13.
14. //Kopierkonstruktor: MyQuaternion(Q)
15. MyQuaternion::MyQuaternion(const MyQuaternion &Q) {
16.     this->dataPtr = new double[4];
17.     for (size_t i =0;i < 4;++i)
18.         this->dataPtr[i] = Q[i];
19.
20. }
21.
22. //Konstruktor mit 4 Argumenten MyQuaternion(e1, e2, e3, e4)
23. MyQuaternion::MyQuaternion(const double& e1,const double& e2,
     const double& e3, const double& e4){
24.     this->dataPtr = new double[4];
25.     this->dataPtr[0] = e1;
26.     this->dataPtr[1] = e2;
27.     this->dataPtr[2] = e3;
28.     this->dataPtr[3] = e4;
29. }
30. //Destructor
31. MyQuaternion::~MyQuaternion(){
32.     delete [] this-> dataPtr;
```

```
33. }
34.
35. //Addition: operator+(QB)
36. MyQuaternion MyQuaternion::operator+
    (const MyQuaternion & QB) const{
37.     MyQuaternion tmp;
38.     for (size_t i =0;i < 4;++i)
39.         tmp.dataPtr[i] = this->dataPtr[i]+QB.dataPtr[i];
40.     return tmp;
41.
42. }
43.
44. //Subtraktion: operator-(QB)
45. MyQuaternion MyQuaternion::operator-
    (const MyQuaternion & QB) const{
46.     MyQuaternion tmp;
47.     for (size_t i =0;i < 4;++i)
48.         tmp.dataPtr[i] = this->dataPtr[i]-QB.dataPtr[i];
49.     return tmp;
50. }
51.
52. //Zugriff auf Komponenten (Setter/Getter): operator[]
53. double & MyQuaternion::operator[]( size_t const n ){
54.     assert(n<4);
55.     return this ->dataPtr[n];
56. }
57.
58. //Zugriff lesend (Getter): operator[] const
59. double& MyQuaternion::operator[]
    ( std::size_t const n ) const {
60.     assert(n<4);
61.     return this->dataPtr[n];
62. }
63.
64. //Kopieroperator: operator=(Q_rhs)
65. MyQuaternion& MyQuaternion::operator= (const MyQuaternion & Q
    _rhs){
66.     for(size_t i=0; i<4; ++i)
67.         this->dataPtr[i]= Q_rhs.dataPtr[i];
68.     return *this;
69. }
70.
71. //
    Zuweisungsoperator mit Skalar als Argument (operator=(skalar)
    )
72. MyQuaternion& MyQuaternion::operator= (const double& c){
73.     for(size_t i=0; i<4; ++i)
74.         this->dataPtr[i]= c;
75.     return *this;
76. }
77.
```

```
78. //conj() const;  =>  gibt konjugierte Quaternion _QA zurück
79. const MyQuaternion MyQuaternion::conj(){
80.     MyQuaternion tmp;
81.     for(size_t i=0; i <3; ++i)
82.         tmp.dataPtr[i]=-this->dataPtr[i];
83.     tmp[3]=this->dataPtr[3];
84.     return tmp;
85. }
86.
87. //
    Berechnung des Produkts zweier Quaternionen: operator*(QB)
88. MyQuaternion MyQuaternion::operator* (const MyQuaternion& QA)
     const{
89.     MyQuaternion tmp;
90.     tmp[0]= this->dataPtr[0]*QA[3]+this->dataPtr[3]*QA[0]-
    this->dataPtr[1]*QA[2]+this->dataPtr[2]*QA[1];
91.     tmp[1]= this->dataPtr[1]*QA[3]+this->dataPtr[3]*QA[1]-
    this->dataPtr[2]*QA[0]+this->dataPtr[0]*QA[2];
92.     tmp[2]= this->dataPtr[2]*QA[3]+this->dataPtr[3]*QA[2]-
    this->dataPtr[0]*QA[1]+this->dataPtr[1]*QA[0];
93.     tmp[3]= this->dataPtr[3]*QA[3]-this->dataPtr[0]*QA[0]-
    this->dataPtr[1]*QA[1]-this->dataPtr[2]*QA[2];
94.     return tmp;
95. }
96.
97. //
    Skalierung aller Komponenten einer Quaternion mit einem Skala
    r: operator*(Skalar)
98. MyQuaternion MyQuaternion::operator* (const double& f) const{

99.     MyQuaternion tmp;
100.     for(size_t i=0; i<4;++i)
101.         tmp.dataPtr[i]= (this->dataPtr[i])*f;
102.     return tmp;
103.}
```

MyQuaternion.h

```
1.  #ifndef MYQUATERNION_H_
2.  #define MYQUATERNION_H_
3.  #include <vector>
4.  #include<cstdlib>
5.
6.
7.
8.  class MyQuaternion {
```

```
9.  public:
10.     //Konstruktor: MyQuaternion()
11.     MyQuaternion();
12.
13.     //Kopierkonstruktor: MyQuaternion(Q)
14.     MyQuaternion(const MyQuaternion &Q);
15.
16.     //
Konstruktor mit 4 Argumenten MyQuaternion(e1, e2, e3, e4)
17.     MyQuaternion(const double& e1,const double& e2, const dou
ble& e3, const double& e4);
18.
19.     //Destructor
20.     ~MyQuaternion();
21.
22.     //Addition: operator+(QB)
23.     MyQuaternion operator+ (const MyQuaternion & QB) const;
24.
25.     //Subtraktion: operator-(QB)
26.     MyQuaternion operator- (const MyQuaternion & QB) const;
27.
28.     //Zugriff auf Komponenten (Setter/Getter): operator[]
29.     double & operator[] ( std::size_t const n );
30.
31.     //Zugriff lesend (Getter): operator[] const
32.     double & operator[] ( std::size_t const n ) const;
33.
34.     //Kopieroperator: operator=(Q_rhs)
35.     MyQuaternion & operator= ( const MyQuaternion & Q_rhs );

36.
37.     //
Zuweisungsoperator mit Skalar als Argument (operator=(skalar)
)
38.     MyQuaternion & operator= ( const double& c );
39.
40.     //
conj() const;  =>  gibt konjugierte Quaternion _QA zurück
41.     const MyQuaternion conj();
42.
43.     //
Berechnung des Produkts zweier Quaternionen: operator*(QB)
44.     MyQuaternion  operator* (const MyQuaternion& QB) const;
45.
46.     //
Skalierung aller Komponenten einer Quaternion mit einem Skala
r: operator*(Skalar)
47.     MyQuaternion  operator* (const double& f) const;
48.
49. private:
50.
```

```
51.     double * dataPtr;
52.
53.
54. };
55.
56. #endif /* MYQUATERNION_H_ */
```

makefile

```
1.  PROG = aufgabe1
2.
3.  FLAGS = -O2 -std=c++11
4.
5.  CC = g++
6.
7.  SRCS = main_aufgabe1.cpp MyQuaternion.cpp
8.
9.  OBJ = $(SRCS:.cpp=.o)
10.
11. all: $(SRCS) $(PROG)
12.
13. $(PROG): $(OBJ)
14.     $(CC) $(FLAGS) $(OBJ) -o $@
15.
16. %.o:%.cpp
17.     $(CC)  $(FLAGS) -c $<
18.
19. clean:
20.     rm -rf *.o $(PROG)
21.
22. ## dependencies
23.
24. MyQuaternion.o: MyQuaternion.cpp MyQuaternion.h
25.
26. main_aufgabe1.o: main_aufgabe1.cpp MyQuaternion.h
```

**Aufgabe2**

MyFunctions.h

```
1.  #ifndef MYFUNCTIONS_H_
2.  #define MYFUNCTIONS_H_
3.
```

```
4.  #include <string>
5.  #include "MyQuaternion.h"
6.
7.  MyQuaternion quaternion_rotation(double x,double y,double z,d
    ouble const theta);
8.  MyQuaternion rotateX(MyQuaternion const &X,MyQuaternion const
     &Q);
9.  void Qprint(const MyQuaternion &,const std::string);
10.
11.
12. #endif /* MYFUNCTIONS_H_ */
```

MyFunctions.cpp

```
1.  #include <cmath>
2.  #include <iostream>
3.  #include "MyFunctions.h"
4.  #define PI acos(-1.0)
5.
6.  using namespace std;
7.
8.  MyQuaternion quaternion_rotation(double x,double y,double z,d
    ouble const theta){
9.
10.   // Quellcode implememtieren
11.     double c =sqrt(pow(x,2.) +pow(y,2.) +pow(z,2.));
12.     MyQuaternion tmp;
13.     tmp[0]=x/c*sin(theta*PI/(180*2));
14.     tmp[1]=y/c*sin(theta*PI/(180*2));
15.     tmp[2]=z/c*sin(theta*PI/(180*2));
16.     tmp[3]=cos(theta*PI/(180*2));
17.     return tmp;
18. }
19.
20. MyQuaternion rotateX(MyQuaternion const &X,MyQuaternion const
     &Q){
21.   // Quellcode implementieren: und auch in Aufgabe3 einfügen

22.     MyQuaternion tmp;
23.     for(auto i =0 ; i< 3;++i)
24.         tmp[i]=-Q[i];
25.     tmp[3]=Q[3];
26.     return Q*X*tmp;
27.
28. }
29.
30.
31. /*
32.  *  vorgegeben:
```

```
 *
 */
void Qprint(const MyQuaternion &Q,const std::string txt="")
{
    cout<<txt<<"("<<Q[0]<<","<<Q[1]<<","<<Q[2]<<","<<Q[3]<<")
"<<endl;
}
```

## Aufgabe3

main_aufgabe3.cpp

```
/*
 * main_aufgabe3.cpp
 *
 *  Created on: 14.12.2019
 *      Author:Yunze Han
 */
// ggf hilfreiche Bibliotheken....
#include <iostream>
#include <fstream>
#include <cmath>
#include <algorithm>
#include <random>
#include <vector>
#include <array>
#include <cassert>
#define PI acos(-1.0)

#include "MyQuaternion.h"
#include "MyFunctions.h"
using namespace std;

// Abkuerzungen fuer Datentypen:
typedef std::array<double,4> Vec4d;

// for seed of random number generator in C++11
std::random_device rd;
std::mt19937 mt(rd());
std::uniform_real_distribution<double> zufall(-1.,1.); //C+
+11 documentation
//http://www.cplusplus.com/reference/random/
uniform_real_distribution/

```

```
31. //Hilfsfunktion fuer random_vec4d:
32. void zufall_in_einheitskreis(double &a, double &b, double &s)
    {
33.     do{
34.         a = zufall(mt);
35.         b = zufall(mt);
36.         s = (a*a+b*b);
37.     }while(s>=1.);
38. }
39.
40. // Algorithm 4 of Marsaglia paper, The Annals of Math. Stat.
    1972,645-6
41. // random point on a surface
42. // hier: Methode 4: point on unit 4d-sphere
43. //
44. void random_vec4d(Vec4d &Q){
45.     double v1,v2,v3,v4,s1,s2;
46.     zufall_in_einheitskreis(v1,v2,s1);
47.     zufall_in_einheitskreis(v3,v4,s2);
48.     //
49.     const double tmp = sqrt(max(0.,(1.-s1))/s2);
50.     Q[0]= v1;
51.     Q[1]= v2;
52.     Q[2]= v3*tmp;
53.     Q[3]= v4*tmp;
54. }
55.
56.
57. // Aufgabe 3.2.A:
58. double get_theta(const MyQuaternion &Q){
59.     // einfügen..
60.     double tmp;
61.     double a= acos(Q[3])*180.0/PI;
62.     //
    double b =asin(sqrt(pow(Q[0],2.)+pow(Q[1],2.)+pow(Q[2],2.)))*
    180.0/PI;
63.     double b =Q[0]*Q[1]*Q[2];
64.     if (a<=90 && a!=0)
65.     {
66.          if(b<0)
67.              {tmp =360. -a;}
68.          else if(b>0)
69.              {tmp =a;}
70.     }
71.     else if(a>90 && a!= 180)
72.     {
73.         if(b>0)
74.         {tmp = a;}
75.         else if(b<0)
76.         {tmp =360. -a;}
77.     }
```

```
78.     else
79.     {
80.         tmp = a;
81.     }
82.
83.     return tmp;
84.
85. }
86.
87. //Aufgabe 3.2.B:
88. double get_theta_cubic(const MyQuaternion &Q){
89.     // einfügen..
90.     MyQuaternion tmp,tmp2,tmp3;
91.
92.     double a,b,c,d,x,y,z,s;
93.
94.
95.     for(auto i=0;i<4;++i){
96.         tmp[i]=abs(Q[i]);
97.     }
98.     if (tmp[0]>=tmp[1]){
99.         a=tmp[0];
100.         b=tmp[1];
101.     }else{
102.         a=tmp[1];
103.         b=tmp[0];
104.     }
105.     if(tmp[2]>=tmp[3]){
106.         c=tmp[2];
107.         d=tmp[3];
108.     }else{
109.         c=tmp[3];
110.         d=tmp[2];
111.     }
112.
113.
114.     if(a>=c){
115.         tmp[3]=a;
116.         if(b>=c){
117.             tmp[2]=b;
118.             tmp[1]=c;
119.             tmp[0]=d;
120.         }else{
121.             tmp[2]=c;
122.             if(b>=d){
123.                 tmp[1]=b;
124.                 tmp[0]=d;
125.             }else{
126.                 tmp[1]=d;
127.                 tmp[0]=b;
128.             }
```

```
        }
    }else{
        tmp[3]=c;
        if(d>=a){
            tmp[2]=d;
            tmp[1]=a;
            tmp[0]=b;
        }else{
            tmp[2]=a;
            if(b>=d){
                tmp[1]=b;
                tmp[0]=d;
            }else{
                tmp[1]=d;
                tmp[0]=b;
            }
        }
    }


    tmp2[0]=(tmp[0]-tmp[1])/sqrt(2.);
    tmp2[1]=(tmp[0]+tmp[1])/sqrt(2.);
    tmp2[2]=(tmp[2]-tmp[3])/sqrt(2.);
    tmp2[3]=(tmp[2]+tmp[3])/sqrt(2.);


    tmp3[0]=(tmp[0]-tmp[1]+tmp[2]-tmp[3])/2.;
    tmp3[1]=(tmp[0]+tmp[1]-tmp[2]-tmp[3])/2.;
    tmp3[2]=(-tmp[0]+tmp[1]+tmp[2]-tmp[3])/2.;
    tmp3[3]=(tmp[0]+tmp[1]+tmp[2]+tmp[3])/2.;


    x=get_theta(tmp);
    y=get_theta(tmp2);
    z=get_theta(tmp3);


    if(x<=y&&x<=z) s=x;
    else if(y<=z&&y<=x) s=y;
    else s=z;

    return s;


}


//
```

```
180.int main(){
181.    //generate data:
182.    Vec4d tmp;
183.    int nmax;
184.
185.    do{
186.        cout<<"Anzahl der Zufallsorientierungen eingeben:";
187.        cin>>nmax;
188.    }while(nmax<1);
189.
190.    std::vector<MyQuaternion> QL(nmax);
191.
192.    //random Initialisierung:
193.    /*....some code
194.     *
195.     */
196.
197.    for (auto i =0 ;i < nmax; ++i)
198.    {
199.        random_vec4d(tmp);
200.        for(auto j=0 ; j< 4;j++)
201.            QL[i][j]=tmp[j];
202.    }
203.
204.
205.
206.
207.    //AUFGABE 3:
208.
209.    //TEIL A)
210.    cout<<"Aufgabe A:"<<endl;
211.    /*some code; calls
212.     *
213.     */
214.    //aufgabe_bearbeiten("random",QL,get_theta);
215.
216.    ofstream Datei;
217.
218.    Datei.open("ergebnis_random.dat",ios::out |
    ios::trunc);
219.
220.    for (auto i =0 ;i < nmax; ++i)
221.        {
222.            //MyQuaternion outme;
223.            for(auto j=0 ; j< nmax;++j)
224.            {
225.                    if(i!=j)
226.                    {
227.                        MyQuaternion outme;
228.                        for(auto k = 0 ;k<3;++k)
229.                        {
```

```
230.                                 outme[k]=-QL[j][k];
231.                             }
232.                             outme[3]=QL[j][3];
233.                             Datei<<get_theta(QL[i]*outme)<<endl;

234.                         }
235.
236.                 }
237.
238.             }
239.     Datei.close();
240.
241.
242.
243.
244.
245.
246.     //TEIL B)
247.     cout<<"Aufgabe B:"<<endl;
248.     /*some code; calls
249.      *
250.      */
251.     // aufgabe_bearbeiten("cube",QL,get_theta_cubic);
252.     Datei.open("ergebnis_cube.dat",ios::out | ios::trunc);
253.
254.         for (auto i =0 ;i < nmax; ++i)
255.             {
256.                 //MyQuaternion outme;
257.                 for(auto j=0 ; j< nmax;++j)
258.                 {
259.                             if(i!=j)
260.                             {
261.                                 MyQuaternion outme;
262.                                 for(auto k = 0 ;k<3;++k)
263.                                 {
264.                                     outme[k]=-QL[j][k];
265.                                 }
266.                                 outme[3]=QL[j][3];
267.                                 Datei<<get_theta_cubic(QL[i]*outm
    e)<<endl;
268.                             }
269.
270.                 }
271.
272.             }
273.         Datei.close();
274.
275.
276.     cout<<"DONE"<<endl;
277.
278.     return 0;
```

```
279.}
```

gethist_random.sh

```
#!/bin/bash


awk '{print (int($0/2)*2+1)}' ./
ergebnis_random.dat > hist_ran.dat


for i in $(seq 1 2 359);
do
{
    echo -n $i;
    echo -n "  ";
    gawk -v nvar=$i '($1==nvar){SUM=SUM+1} END{print SUM/
NR}' hist_ran.dat

}>>hist_random.dat

done
```

gethist_cube.sh

```
#!/bin/bash


awk '{print ((int($0*5)*2+1)/10)}' ./
ergebnis_cube.dat > hist_cu.dat

for i in $(seq 0.1 0.2 45.1);
do
{
    echo -n $i;
    echo -n "  ";
    gawk -v nvar=$i '($1==nvar){SUM=SUM+1} END{print SUM/
NR}' hist_cu.dat

}>>hist_cube.dat

done
```