EasyChair Preprint

№ 6349

Maple Program for MC 2021 Paper: a Machine Proof of an Inequality for the Sum of Distances between Four Points on the Unit Hemisphere using Maple Software

Zhenbing Zeng, Yaochen Xu, Jian Lu and Liangyu Chen

August 23, 2021

Maple Program for MC2021 Paper "A Machine Proof of an Inequality for the Sum of Distances between Four Points on the Unit Hemisphere using Maple Software" *

Zhenbing Zeng[1][0000−0002−9728−1114], Yaochen Xu[1,2], Jian Lu[1], and Liangyu Chen[3]

[1] Department of Mathematics, Shanghai University, Shanghai 200444, China
zbzeng@shu.edu.cn, lujian@picb.ac.cn
[2] SIBCB, Chinese Academy of Sciences, Shanghai 200031, China
xuyaochen@sibcb.ac.cn
[3] Software Engineering Institute, East China Normal University, Shanghai 200062, China, lychen@sei.ecnu.edu.cn

Abstract. In this document, we present the Maple program for our Maple Conference 2021 paper, where we proved a geometrical inequality which states that for any four points on the unit hemisphere, the largest sum of distances between the points is $4+4\sqrt{2}$ using Maple computation. In our proof, we have constructed a rectangular neighborhood of the local maximum point in the feasible set, which size is explicitly determined, and proved that (1): the objective function is bounded by a quadratic polynomial which takes the local maximum point as the unique critical point in the neighbor- hood, and (2): the rest part of the feasible set can be partitioned into a finite union of a large number of very small cubes so that on each small cube the conjecture can be verified by estimating the objective function with exact numerical computation. The attched Maple program is for the second part. The work on first part, i.e., the construction of the critical neighborhood, has been published recently in the ADG 2021 (the Thirteenth International Conference on Automated Deduction in Geometry), where we have proved that the sum of distances between points contained in the constructed neighborhoods is not larger than $4 + 4\sqrt{2}$, also using Maple computatin.

Keywords: computational geometry, inequality, global search algorithm, branch and bound, Maple program

* This work was supported by NSFC under Grant Nos. 61772203 and 12071282.

Part 1: Maple Program for Numerical Search

```
###1
construct256squares := proc() local i, j, sqs, x, y;
sqs := [];
for i from 0 to 15 do
for j from 0 to 15 do
x := -15/16 + 1/8*i; y := -15/16 + 1/8*j;
sqs := [op(sqs), [x, y, 1/8, i, j]];
end do;
end do;
end proc;
construct256squares(); nops(%);
                              256

###2
construct224squares := proc() local i, j, sqs, x, y, rs;
sqs := [];
for i from 0 to 15 do
for j from 0 to 15 do
x := -15/16 + 1/8*i; y := -15/16 + 1/8*j;
rs := [(x + 1/16)^2 + (y + 1/16)^2,
(x + 1/16)^2 + (y - 1/16)^2,
(x - 1/16)^2 + (y + 1/16)^2,
(x - 1/16)^2 + (y - 1/16)^2];
if min(op(rs)) <= 1 then
sqs := [op(sqs), [x, y, 1/8, i, j]];
end if;
end do;
end do;
sqs;
end proc;

construct224squares(); nops(%);
                              224

###3
construct60squares := proc() local i, j, sqs, x, y, rs;
sqs := [];
for i from 0 to 15 do
for j from 0 to 15 do
x := -15/16 + 1/8*i; y := -15/16 + 1/8*j;
rs := [(x + 1/16)^2 + (y + 1/16)^2,
(x + 1/16)^2 + (y - 1/16)^2,
(x - 1/16)^2 + (y + 1/16)^2,
```

```
(x - 1/16)^2 + (y - 1/16)^2];
if min(op(rs)) <= 1 and 1 <= max(op(rs)) then
sqs := [op(sqs), [x, y, 1/8, i, j]];
end if;
end do;
end do;
sqs;
end proc;

construct60squares(); nops(%);
                                60

###4
squareb := proc() local sq60, i, x, y, b, sqb, bd2;
sq60 := construct60squares();
b := 1/8;
bd2 := 1.9331^2;
sqb := [];
for i to nops(sq60) do
x := op(1, op(i, sq60)); y := op(2, op(i, sq60));
if 0 <= x and 0 <= y and 2 + 2*y < bd2 then
sqb := [op(sqb), op(i, sq60)];
end if;
end do;
sqb;
end proc;

squareb(); nops(%);
                                11

###5
squarec := proc() local sq60, i, x, y, b, sqc, bd1;
sq60 := construct60squares();
b := 1/8;
bd1 := 0.992^2;
sqc := [];
for i to nops(sq60) do
x := op(1, op(i, sq60)); y := op(2, op(i, sq60));
if x < 0 and bd1 < 17/8 + 2*y and 2*y <= 7/8 then
sqc := [op(sqc), op(i, sq60)];
end if;
end do;
sqc;
end proc;
```

```
squarec(); nops(%);
                                11

###6
squared102 := proc() local sq224, i, x, y, b, sqd, bd1, bd2;
sq224 := construct224squares();
b := 1/8;
bd1 := 0.992^2;
bd2 := 0.6764^2;
sqd := [];
for i to nops(sq224) do
x := op(1, op(i, sq224)); y := op(2, op(i, sq224));
if bd1 < 17/8 + 2*y and sqheight2(op(i, sq224)) < bd2 then
sqd := [op(sqd), op(i, sq224)];
end if;
end do;
sqd;
end proc;

###7
sqheight2 := proc(square) local x, y, b, hts;
x := op(1, square); y := op(2, square); b := op(3, square);
hts := [1 - (x + 1/2*b)^2 - (y + 1/2*b)^2,
1 - (x + 1/2*b)^2 - (y - 1/2*b)^2,
1 - (x - 1/2*b)^2 - (y + 1/2*b)^2,
1 - (x - 1/2*b)^2 - (y - 1/2*b)^2];
hts := min(op(hts));
end proc;

squared102(); nops(%);
                               102

###8
warp2betweensquares := proc(square1, square2)
local x1, y1, b1, x2, y2, b2, ws, i1, j1, i2, j2;
x1 := op(1, square1); y1 := op(2, square1); b1 := op(3, square1);
x2 := op(1, square2); y2 := op(2, square2); b2 := op(3, square2);
b1 := 1/2*b1; b2 := 1/2*b2;
ws := [];
for i1 in [-1, 1] do
for j1 in [-1, 1] do
for i2 in [-1, 1] do
for j2 in [-1, 1] do
ws := [op(ws),
2 - (2*x1 + 2*i1*b1)*(x2 + i2*b2) - (2*y1 + 2*j1*b1)*(y2 + j2*b2)];
```

```
end do;
end do;
end do;
end do;
max(op(ws));
end proc;

warp2betweensquares(op(110, construct256squares()),
op(151, construct256squares()));

###9
squarebnear := proc() local sqb, sqn, sq224, i, bd1;
sq224 := construct224squares();
sqb := squareb();
bd1 := 0.992^2;
sqn := [];
for i to nops(sq224) do
if max(op(map(proc(sqx) warp2betweensquares(op(i, sq224), sqx); end proc, sqb))) < bd1 then
sqn := [op(sqn), op(i, sq224)];
end if;
end do;
sqn;
end proc;

squarebnear(); nops(%);
                                9

###10
squarecnear := proc() local sqc, sqn, sq224, i, bd1;
sq224 := construct224squares();
sqc := squarec();
bd1 := 0.992^2;
sqn := [];
for i to nops(sq224) do
if max(op(map(proc(sqx) warp2betweensquares(op(i, sq224), sqx); end proc, sqc))) < bd1 then
sqn := [op(sqn), op(i, sq224)];
end if;
end do;
sqn;
end proc;

squarecnear(); nops(%);
                                3

###11
```

```
squaredanear := proc() local sq102, sqa, i, pta, bd2;
bd2 := 0.7256^2;
sq102 := squared102();
sqa := [];
pta := [0, -1];
for i to nops(sq102) do
if maxdistance2pointtosquare(pta, op(i, sq102)) < bd2 then
sqa := [op(sqa), op(i, sq102)];
end if;
end do;
sqa;
end proc;

###12
maxdistance2pointtosquare := proc(point, square) local x0, y0, x1, y1, b, dst2;
x0 := op(1, point); y0 := op(2, point);
x1 := op(1, square); y1 := op(2, square);
b := 1/2*op(3, square);
dst2 := [(x0 - x1 + b)^2 + (y0 - y1 + b)^2,
(x0 - x1 + b)^2 + (y0 - y1 - b)^2,
(x0 - x1 - b)^2 + (y0 - y1 + b)^2,
(x0 - x1 - b)^2 + (y0 - y1 - b)^2];
max(dst2);
end proc;

squaredanear(); nops(%);
                                2

###13
squared := proc() local sq102, sqxd, i, j, k, sqd;
sq102 := squared102();
sqxd := [op(squarebnear()), op(squarecnear()), op(squaredanear())];
[nops(sq102), nops(sqxd)];
sqd := [];
for i to nops(sq102) do
k := 0;
for j to nops(sqxd) do
if op(i, sq102) = op(j, sqxd) then
k := k + 1;
end if;
end do;
if k = 0 then
sqd := [op(sqd), op(i, sq102)];
end if;
end do;
```

```
sqd;
end proc;

squared(); nops(%);
                                88

###14
sqb11 := squareb();
sqc11 := squarec();
sqd88 := squared();
map(nops, [sqb11, sqc11, sqd88]);
                          [11, 11, 88]

###15
kaifang := proc(a) local rx, x1, x2, x;
x1 := 0;
x2 := abs(2*a);
while 1/100000000 < abs(x2 - x1) do
x := 1/2*x1 + 1/2*x2;
if a < x^2 then
x2 := x; else
x1 := x;
end if;
end do;
x2;
end proc;

kaifang(2);
                           47453133/33554432
evalf(%);
               1.41421356797218322753906250000
evalf(2^(1/2));
               1.41421356237309504880168872421

###16
testquadrat8 := proc(sb, sc, sd)
local xb, yb, xc, yc, xd, yd, d2ab, d2ac, d2ad, w2bc, w2cd, w2db, dw6;
xb := op(1, sb); yb := op(2, sb);
xc := op(1, sc); yc := op(2, sc);
xd := op(1, sd); yd := op(2, sd);
d2ab := 19/9 + 2*yb;
d2ac := 17/8 + 2*yc;
d2ad := 17/8 + 2*yd;
w2bc := warp2betweensquares(sb, sc);
w2cd := warp2betweensquares(sc, sd);
```

```
w2db := warp2betweensquares(sd, sb);
dw6 := map(kaifang, [d2ab, d2ac, d2ad, w2bc, w2cd, w2db]);
if 9.65685424949 < addlist(dw6) then
1; else
0;
end if;
end proc;

###17
addlist := proc(list) local i, sm;
sm := 0;
for i to nops(list) do
sm := sm + op(i, list);
end do;
sm;
end proc;

evalf(4 + 4*2^(1/2));
                9.656854249492380195206754896840
Digits := 30;
                          Digits := 30

###18
refinequadrats8 := proc() local sqb, sqc, sqd, remains, i, j, k;
remains := [];
for i to nops(sqb11) do
sqb := op(i, sqb11);
for j to nops(sqc11) do
sqc := op(j, sqc11);
for k to nops(sqd88) do
sqd := op(k, sqd88);
if testquadrat8(sqb, sqc, sqd) = 1 then
remains := [op(remains), [i, j, k]];
end if;
end do;
end do;
end do;
remains;
end proc;

time1 := time();
poq4300 := refinequadrats8();
nops(%);
time() - time1;
                        time1 := 15314.625
```

```
                                4300
                               14.531

testquadrat8(op(1, sqb11), op(4, sqc11), op(70, sqd88));
                                 1
22^3 - 4300; %/22^3;
                                6348
                                1587/2662
evalf(%);
                0.59616829451540195341848234410 2

###19
square16division := proc(square) local x, y, b, i, j, rst;
x := op(1, square); y := op(2, square); b := op(3, square);
rst := [];
for i from 0 to 3 do
for j from 0 to 3 do
rst := [op(rst), [x + 1/8*(2*i - 3)*b,
y + 1/8*(2*j - 3)*b, 1/4*b,
op(4 .. -1, square), 4*i + j]];
end do;
end do;
rst;
end proc;

square16division([-1/2, 1/2, 1, 0]);
op(3, sqb11);
square16division(%);

###20
checksquareintersectcircle := proc(square) local x, y, b, ds2;
x := op(1, square); y := op(2, square); b := 1/2*op(3, square);
ds2 := [(x - b)^2 + (y - b)^2,
(x - b)^2 + (y + b)^2,
(x + b)^2 + (y - b)^2,
(x + b)^2 + (y + b)^2];
if 1 <= max(op(ds2)) and min(op(ds2)) <= 1 then
1; else
0;
end if;
end proc;

###21
chesquareintersectdisk := proc(square) local x, y, b, ds2;
x := op(1, square); y := op(2, square); b := 1/2*op(3, square);
```

```
ds2 := [(x - b)^2 + (y - b)^2,
(x - b)^2 + (y + b)^2,
(x + b)^2 + (y - b)^2,
(x + b)^2 + (y + b)^2];
if min(op(ds2)) <= 1 then
1; else
0;
end if;
end proc;

###22
testbc11div16 := proc(sqlist) local i, div16, result;
div16 := [];
for i to nops(sqlist) do
div16 := [op(div16), op(square16division(op(i, sqlist)))];
end do;
nops(div16);
result := [];
for i to nops(div16) do
if checksquareintersectcircle(op(i, div16)) = 1 then
result := [op(result), op(i, div16)];
end if;
end do;
result;
end proc;

testbc11div16(sqd88); nops(%);
                              140

###23
testd88intersectcircle := proc() local i, rst;
rst := [];
for i to nops(sqd88) do
if checksquareintersectcircle(op(i, sqd88)) = 1 then
rst := [op(rst), op(i, sqd88)];
end if;
end do;
rst;
end proc;

testd88intersectcircle();
nops(%);
                               32

###24
```

```
testd88div16 := proc() local i, j, d88div16, sqi, result;
d88div16 := testd88intersectcircle();
result := [];
for i to nops(d88div16) do
sqi := op(i, d88div16);
sqi := square16division(sqi);
for j to nops(sqi) do
if chesquareintersectdisk(op(j, sqi)) = 1 then
result := [op(result), op(j, sqi)];
end if;
end do;
end do;
nops(result);
end proc;

testd88div16();
                              398
16*32;
                              512
398/512;
evalf(%);
               0.77734375000000000000000000000
398/32;
evalf(%);
               12.43750000000000000000000000000

###25
squarebnotfartopointa := proc(square) local x, y, b;
x := op(1, square); y := op(2, square); b := op(3, square);
if 2 + 2*y - b < 1.9331^2 then
1; else
0;
end if;
end proc;

###26
squarecnotclosetopointa := proc(square) local x, y, b;
x := op(1, square); y := op(2, square); b := op(3, square);
if 0.992^2 < 2 + 2*y + b then
1; else
0;
end if;
end proc;

###27
```

```
squarednotclosetopointa := proc(square) local x, y, b;
x := op(1, square); y := op(2, square); b := op(3, square);
if 0.992^2 < 2 + 2*y + b
and 0.7256^2 < (abs(x) + 1/64)^2 + (-65/64 - y)^2
and 1 + (-1)*0.6764^2 <= (abs(x) + 1/64)^2 + (abs(y) + 1/64)^2 then
1; else
0;
end if;
end proc;

###28
squarednottoohigh := proc(square) local x, y, b;
x := op(1, square); y := op(2, square); b := op(3, square);
if 1 + (-1)*0.6764^2 <= (abs(x) + 1/64)^2 + (abs(y) + 1/64)^2 then
1; else
0;
end if;
end proc;

map(squarebnotfartopointa, construct60squares());
[1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
  1, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 1, 1, 0, 1, 1,
  1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1]
addlist(%);
                                52
% - 30;
                                22
%/2;
                                11

map(squarecnotclosetopointa, construct60squares());
[1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 0, 0, 1, 1, 0, 1, 0,
  0, 1, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 0, 1, 1, 0, 1,
  0, 0, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1]
addlist(%);
                                42
% - 30;
                                12
%/2;
                                6

map(squarednotclosetopointa, squared102());
[1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
  1, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 0, 0, 1, 1, 1, 0, 0, 1, 1, 1,
  1, 1, 1, 0, 1, 1, 0, 1, 1, 0, 1, 1, 0, 1, 1, 1, 1, 1, 0, 0, 1,
```

```
  1, 1, 1, 0, 0, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1,
  1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1]
nops(%);
                              102
addlist(`%%`);
                               82

map(squarednotclosetopointa, construct224squares());
[1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0,
  0, 1, 1, 1, 0, 0, 0, 0, 1, 1, 1, 1, 1, 0, 0, 1, 0, 0, 0, 0, 0,
  0, 0, 0, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1,
  1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 0, 0, 0, 0,
  0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0,
  0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
  1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0,
  0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0,
  0, 0, 0, 0, 0, 1, 1, 1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1,
  1, 0, 0, 1, 1, 1, 0, 0, 0, 0, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1,
  1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1]
addlist(%);
                               82
nops(`%%`) - %;
                              142

###29
quadratnearsingular := proc(sqbb, sqcc, sqdd) local xb, yb, xc, yc, xd, yd, b;
xb := op(1, sqbb); yb := op(2, sqbb);
xc := op(1, sqcc); yc := op(2, sqcc);
xd := op(1, sqdd); yd := op(2, sqdd);
b := op(3, sqbb);
b := 1/32;
if 24/25 <= xb - 1/2*b
and -7/25 <= yb - 1/2*b
and yb + 1/2*b <= 7/25
and xc + 1/2*b <= -24/25
and -7/25 <= yc - 1/2*b
and yc + 1/2*b <= 7/25
and -7/25 <= xd - 1/2*b
and xd + 1/2*b <= 7/25
and 24/25 <= yd - 1/2*b
then
1; else
0;
end if;
end proc;
```

```
###30
testb11div16 := proc() local i, div16, result;
div16 := [];
for i to nops(sqb11) do
div16 := [op(div16), op(square16division(op(i, sqb11)))];
end do;
nops(div16);
result := [];
for i to nops(div16) do
if checksquareintersectcircle(op(i, div16)) = 1
and squarebnotfartopointa(op(i, div16)) = 1
then
result := [op(result), op(i, div16)];
end if;
end do;
result;
end proc;

testb11div16();
nops(%);
                                44

###31
testc11div16 := proc() local i, div16, result;
div16 := [];
for i to nops(sqc11) do
div16 := [op(div16), op(square16division(op(i, sqc11)))];
end do;
nops(div16);
result := [];
for i to nops(div16) do
if checksquareintersectcircle(op(i, div16)) = 1
and squarednotclosetopointa(op(i, div16)) = 1
then
result := [op(result), op(i, div16)];
end if;
end do;
result;
end proc;

testc11div16();
nops(%);
                                41
```

```
square16division([-13/16, -7/16, 1/8]);
map(proc(sqx) if checksquareintersectcircle(sqx) = 1 then sqx; else 0; end if; end proc, %)
2 + 2*(-31/64 + 1/64);
                                17/16

###32
psidivisionb := proc(square) local i, sqs16, rst;
sqs16 := square16division(square);
rst := [];
for i to nops(sqs16) do
if checksquareintersectcircle(op(i, sqs16)) = 1
and (1 = 0 or squarebnotfartopointa(op(i, sqs16)) = 1)
then
rst := [op(rst), op(i, sqs16)];
end if;
end do;
rst;
end proc;

###33
psidivisionc := proc(square) local i, sqs16, rst;
sqs16 := square16division(square);
rst := [];
for i to nops(sqs16) do
if checksquareintersectcircle(op(i, sqs16)) = 1
and (1 = 0 or squarednotclosetopointa(op(i, sqs16)) = 1)
then
rst := [op(rst), op(i, sqs16)];
end if;
end do;
rst;
end proc;

###34
phidivisiond := proc(square) local i, sqs16, rst;
sqs16 := square16division(square);
rst := [];
for i to nops(sqs16) do
if chesquareintersectdisk(op(i, sqs16)) = 1
and (1 = 0 or squarednotclosetopointa(op(i, sqs16)) = 1)
then
rst := [op(rst), op(i, sqs16)];
end if;
end do;
rst;
```

```
end proc;

map(psidivisionb, sqb11);
map(nops, %);
               [1, 7, 5, 2, 1, 7, 2, 4, 5, 5, 5]
[1, 7, 5, 2, 1, 7, 2, 4, 5, 5, 5];
[1, 7, 5, 2, 1, 7, 2, 4, 5, 5, 5];
map(psidivisionc, sqc11);
map(nops, %);
               [5, 5, 5, 4, 4, 5, 5, 5, 1, 1, 1]
addlist(%);
                                41
[5, 5, 5, 4, 4, 5, 5, 5, 1, 1, 1];
[5, 5, 5, 4, 4, 5, 5, 5, 7, 1, 1];
square16division(%);
map(squarednotclosetopointa, %);
        [0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 1]
map(phidivisiond, sqd88);
map(nops, %);
add(%);
[6, 12, 16, 12, 6, 4, 16, 16, 16, 16, 16, 16, 16, 16, 13, 2, 4,
  16, 14, 7, 4, 4, 7, 14, 16, 16, 15, 2, 4, 8, 8, 16, 16, 13, 8,
  16, 16, 6, 14, 16, 12, 7, 16, 16, 4, 16, 16, 4, 16, 16, 7, 16,
  16, 14, 16, 12, 8, 16, 16, 6, 4, 8, 8, 16, 16, 13, 4, 16, 14,
  7, 4, 4, 7, 14, 4, 16, 16, 16, 16, 16, 16, 16, 6, 12, 16, 16,
  16, 16]
                              1042
addlist(%);
                              1294

###35
quadrat32statics := proc() local i, j, k, rs; rs := 0;
for i to nops(sqb11) do
for j to 11 do
for k to nops(sqd88) do
rs := rs + nops(psidivisionb(op(i, sqb11)))
*nops(psidivisionc(op(j, sqc11)))
*nops(phidivisiond(op(k, sqd88)));
end do;
end do;
end do;
rs;
end proc;

quadrat32statics();
```

```
                              1879768
(16*4*4)*4300;
2215312;
2675992;
1879768;
                              1100800
                              2215312
                              2675992
                              1879768
[2215312/2675992, 1879768/2675992];
evalf(%);
              [0.827847018974645664112598243941,

                0.702456509585977835509224242823]

###36
takequadratsnearsingular := proc() local i, j, k, sqsbb, sqscc, sqsdd, i1, j1, k1, result;
result := [];
for i to nops(sqb11) do
sqsbb := psidivisionb(op(i, sqb11));
for j to nops(sqc11) do
sqscc := psidivisionc(op(j, sqc11));
for k to nops(sqd88) do
sqsdd := phidivisiond(op(k, sqd88));
if 1 <= nops(sqsbb) and 1 <= nops(sqscc) and 1 <= nops(sqsdd)  then
for i1 to nops(sqsbb) do
for j1 to nops(sqscc) do
for k1 to nops(sqsdd) do
if quadratnearsingular(op(i1, sqsbb), op(j1, sqscc), op(k1, sqsdd)) = 1
then
result := [op(result), 1];
end if;
end do;
end do;
end do;
end if;
end do;
end do;
end do;
result;
end proc;

eq3 := proc(list1, list2)
op(1, list1) = op(1, list2)
and op(2, list1) = op(2, list2)
```

```
and op(3, list1) = op(3, list2);
end proc;

takequadratsnearsingular();
nops(%);
                                2048
(24/25 + 1/64) + 1/64;
                                793/800

square16division([15/16, 1/16, 1/8]);
square16division([-15/16, 1/16, 1/8]);
square16division([1/16, 15/16, 1/8]);

[[63/64, 1/64, 1/32, 12], [-63/64, 1/64, 1/32, 0], [1/64, 63/64, 1/32, 3]];
quadratnearsingular(op(%));
                                 1
op(8, sqb11);
square16division(%);
op(5, sqc11);

###37
findsquare := proc(sq1, sqlist) local i, rst;
rst := 0;
for i to nops(sqlist) do
if op(1, op(i, sqlist)) = op(1, sq1)
and op(2, op(i, sqlist)) = op(2, sq1)
and op(3, op(i, sqlist)) = op(3, sq1)
then
rst := i;
end if;
end do;
rst;
end proc;

findsquare([1/16, 15/16, 1/8], sqd88);
                                 50
nops(poq4300);
                                4300

###38
counting32 := proc() local n1, i, j, k, sqbb, sqcc, sqdd, rst;
rst := 0;
for n1 to nops(poq4300) do
i := op(1, op(n1, poq4300));
j := op(2, op(n1, poq4300));
```

```
k := op(3, op(n1, poq4300));
sqbb := psidivisionb(op(i, sqb11));
sqcc := psidivisionc(op(j, sqc11));
sqdd := phidivisiond(op(k, sqd88));
rst := rst + nops(sqbb)*nops(sqcc)*nops(sqdd);
end do;
rst;
end proc;

counting32();
844917;
1105782;
844917/1105782;
evalf(%);
                              16567/21682
                0.76409002859514804907296374873 2

###39
counting32s := proc() local n1, i, j, k, sqbb, sqcc, sqdd, rst, i1, j1, k1;
rst := 0;
for n1 to nops(poq4300) do
i := op(1, op(n1, poq4300));
j := op(2, op(n1, poq4300));
k := op(3, op(n1, poq4300));
sqbb := psidivisionb(op(i, sqb11));
sqcc := psidivisionc(op(j, sqc11));
sqdd := phidivisiond(op(k, sqd88));
if 0 < nops(sqbb)*nops(sqcc)*nops(sqdd) then
for i1 to nops(sqbb) do
for j1 to nops(sqcc) do
for k1 to nops(sqdd) do
if quadratnearsingular(op(i1, sqbb), op(j1, sqcc), op(k1, sqdd)) = 1 then
rst := rst + 1;
end if;
end do;
end do;
end do;
end if;
end do;
rst;
end proc;

counting32s();
                                2048
```

```
###40
checkwarpdistancesum := proc(square1, square2, square3)
local xb, yb, xc, yc, xd, yd, d2ab, d2ac, d2ad, wd2bc, wd2cd, wd2db, sm, bd2;
bd2 := 9.65685424949;
xb := op(1, square1); yb := op(2, square1);
xc := op(1, square2); yc := op(2, square2);
xd := op(1, square3); yd := op(2, square3);
d2ab := 65/32 + 2*yb;
d2ac := 65/32 + 2*yc;
d2ad := 65/32 + 2*yd;
wd2bc := g32(xb, yb, xc, yc, 1/32);
wd2cd := g32(xc, yc, xd, yd, 1/32);
wd2db := g32(xd, yd, xb, yb, 1/32);
sm := map(kaifang, [d2ab, d2ac, d2ad, wd2bc, wd2cd, wd2db]);
sm := addlist(sm);
if bd2 < sm then
1; else
0;
end if;
end proc;

###41
g32 := proc(u1, v1, u2, v2, b) local g0, g12, ksi1, ksi2, ita1, ita2;
g0 := 2 - 2*u1*u2 - 2*v1*v2;
g12 := [];
for ksi1 in [-1, 1] do
for ita1 in [-1, 1] do
for ksi2 in [-1, 1] do
for ita2 in [-1, 1] do
g12 := [op(g12),
b*(ksi1*u1 + ita1*v1 + ksi2*u2 + ita2*v2)
+ b^2*(ksi2*ksi1 + ita1*ita2)];
end do;
end do;
end do;
end do;
g12 := max(g12);
g0 + g12;
end proc;

###42
counting32sm := proc()
local n1, i, j, k, sqbb, sqcc, sqdd, rst, i1, j1, k1, sqbbi1, sqccj1, sqddk1, remains;
rst := 0;
remains := [];
```

```
for n1 to nops(poq4300) do
i := op(1, op(n1, poq4300));
j := op(2, op(n1, poq4300));
k := op(3, op(n1, poq4300));
sqbb := psidivisionb(op(i, sqb11));
sqcc := psidivisionc(op(j, sqc11));
sqdd := phidivisiond(op(k, sqd88));
if 0 < nops(sqbb)*nops(sqcc)*nops(sqdd) then
for i1 to nops(sqbb) do
for j1 to nops(sqcc) do
for k1 to nops(sqdd) do
sqbbi1 := op(i1, sqbb);
sqccj1 := op(j1, sqcc);
sqddk1 := op(k1, sqdd);
if quadratnearsingular(op(i1, sqbb), op(j1, sqcc), op(k1, sqdd)) = 1 then
rst := rst + 1;
else
if checkwarpdistancesum(op(i1, sqbb), op(j1, sqcc), op(k1, sqdd)) = 1 then  remains := [op(
end if;
end if;
end do;
end do;
end do;
end if;
end do;
[rst, remains];
end proc;

time1 := time();
snoq21254 := counting32sm();
nops(op(2, snoq21254));
time() - time1;
                      time1 := 3078.859
                            21254
                           1170.266
21254/844917;
evalf(%);
              0.025155133581168327765745037678
844917 - 21254;
                             823663
evalf(%);
                             823663/844917
                0.97484486641883167222342549623 2
4 + 4*sqrt(2);
                                 (1/2)
```

```
                          4 + 4 2
evalf(%);
                9.65685424949238019520675489684
sqb11;
op(100, op(2, snoq21254));

###43
deepsearch := proc(quadrat)
local tasklist, firsttask, firsttaskresult, newtask, workstime, depth;
tasklist := constructtask(quadrat);
workstime := 0;
depth := op(3, op(1, quadrat));
while `not`(tasklist = []) do
firsttask := op(1, tasklist);
depth := min(depth, op(3, op(1, firsttask)));
firsttaskresult := dothework(firsttask);
if firsttaskresult = 1 then
tasklist := [op(2 .. nops(tasklist), tasklist)];
else
newtask := constructtask(firsttask);
tasklist := [op(newtask), op(2 .. nops(tasklist), tasklist)];
end if;
firsttaskresult := dowork(firstwork);
workstime := workstime + 1;
end do;
[workstime, depth];
depth;
end proc;

###44
constructtask := proc(quadrat1)
local x, y, b, i, j, k, sqb, sqc, sqd, sqsbb, sqscc, sqsdd, corner4, quadrats;
sqb := op(1, quadrat1);
sqc := op(2, quadrat1);
sqd := op(3, quadrat1);
x := op(1, sqb);
y := op(2, sqb);
b := op(3, sqb);
sqb := [];
for i from 0 to 3 do
for j from 0 to 3 do
sqb := [op(sqb), [x + 1/2*(-3/4 + 1/2*i)*b, y + 1/2*(-3/4 + 1/2*j)*b, 1/4*b]];
end do;
end do;
x := op(1, sqc);
```

```
y := op(2, sqc);
b := op(3, sqc);
sqc := [];
for i from 0 to 3 do
for j from 0 to 3 do
sqc := [op(sqc), [x + 1/2*(-3/4 + 1/2*i)*b, y + 1/2*(-3/4 + 1/2*j)*b, 1/4*b]];
end do;
end do;
x := op(1, sqd);
y := op(2, sqd);
b := op(3, sqd);
sqd := [];
for i from 0 to 3 do
for j from 0 to 3 do
sqd := [op(sqd), [x + 1/2*(-3/4 + 1/2*i)*b, y + 1/2*(-3/4 + 1/2*j)*b, 1/4*b]];
end do;
end do;
[sqb, sqc, sqd];
sqsbb := [];
for k to nops(sqb) do
x := op(1, op(k, sqb));
y := op(2, op(k, sqb));
b := op(3, op(k, sqb));
corner4 := [];
for i in [-1, 1] do
for j in [-1, 1] do
corner4 := [op(corner4), (x - 1/2*i*b)^2 + (y - 1/2*j*b)^2];
end do;
end do;
if 1 <= max(op(corner4)) and min(op(corner4)) <= 1 then
sqsbb := [op(sqsbb), op(k, sqb)];
end if;
end do;
sqscc := [];
for k to nops(sqc) do
x := op(1, op(k, sqc));
y := op(2, op(k, sqc));
b := op(3, op(k, sqc));
corner4 := [];
for i in [-1, 1] do
for j in [-1, 1] do
corner4 := [op(corner4), (x - 1/2*i*b)^2 + (y - 1/2*j*b)^2];
end do;
end do;
if 1 <= max(op(corner4)) and min(op(corner4)) <= 1 then
```

```
sqscc := [op(sqscc), op(k, sqc)];
end if;
end do;
sqsdd := [];
for k to nops(sqd) do
x := op(1, op(k, sqd));
y := op(2, op(k, sqd));
b := op(3, op(k, sqd));
corner4 := [];
for i in [-1, 1] do
for j in [-1, 1] do
corner4 := [op(corner4), (x - 1/2*i*b)^2 + (y - 1/2*j*b)^2];
end do;
end do;
if min(op(corner4)) <= 1 then
sqsdd := [op(sqsdd), op(k, sqd)];
end if;
end do;
[sqsbb, sqscc, sqsdd];
quadrats := [];
for i to nops(sqsbb) do
for j to nops(sqscc) do
for k to nops(sqsdd) do
quadrats := [op(quadrats), [op(i, sqsbb), op(j, sqscc), op(k, sqsdd)]];
end do;
end do;
end do;
quadrats;
end proc;

constructtask(op(21254, op(2, snoq21254)));
nops(%);
                                96

###45
dothework := proc(task1)
local xb, yb, b, xc, yc, xd, yd, ab, ac, ad, bc, cd, db, g2, opt, wd6;
opt := 4 + 4*2^(1/2);
opt := 9.65685424;
xb := op(1, op(1, task1));
yb := op(2, op(1, task1));
b := op(3, op(1, task1));
xc := op(1, op(2, task1));
yc := op(2, op(2, task1));
xd := op(1, op(3, task1));
```

```
yd := op(2, op(3, task1));
ab := 2 + 2*yb + b;
ac := 2 + 2*yc + b;
ad := 2 + 2*yd + b;
bc := 2 - 2*xb*xc - 2*yb*yc + b*(abs(xb) + abs(yb) + abs(xc) + abs(yc)) + b^2;
cd := 2 - 2*xd*xc - 2*yc*yd + b*(abs(xc) + abs(yc) + abs(xd) + abs(yd)) + b^2;
db := 2 - 2*xd*xb - 2*yd*yb + b*(abs(xd) + abs(yd) + abs(xb) + abs(yb)) + b^2;
wd6 := kaifang(ab) + kaifang(ac) + kaifang(ad) + kaifang(bc) + kaifang(cd) + kaifang(db);
if wd6 < opt then
1; else
0;
end if;
end proc;

###46
checkxxx := proc(aaa, bbb, xxx) local ccc, k;
ccc := [];
for k to xxx do
ccc := [op(ccc), eq3(op(k, aaa), op(k, bbb))];
end do;
ccc;
end proc;

4 + 4*2^(1/2);
evalf(%);
                9.656854249492380195206754896840
op(13000, op(2, snoq21254));
deepsearch(%);
                                1/128
###37
testdeepsearch := proc(k) local i, result, t1;
t1 := time();
result := [];
for i to min(k, nops(op(2, snoq21254))) do
result := [op(result), deepsearch(op(i, op(2, snoq21254)))];
end do;
print(time() - t1);
min(op(result));
end proc;

testdeepsearch(21254);
8777.250;
1/512;
```

Part 2: Maple Program for Local Critical Analysis

This part of Maple program is saved in my ThinkPad notebook computer, in the following place:

`C:\Users\Administrator\documents\hagenberg-0218.mw`

For explanation to the local critical analysis, see [1], [2] and [3].

References

1. Zeng, Z., Yang, Z.. How large is the locally optimal region of a locally-optimal point? International Workshop on Certified and Reliable Computation Nanning, Guangxi, China, July 17-20, 2011.
2. Zeng, Z., Zhang, J.. A Mechanical Proof to a Geometric Inequality of Zirakzadeh Through Rectangular Partition of Polyhedra (in Chinese). Journal of Systems Science and Complexity, 2010, 30(11): 1430-1458.
3. Zeng, Z., Lu, J., Xu, Y., Wang, Y.. Maximizing the Sum of the Distances between Four Points on the Unit Hemisphere Proc. ADG 2021 (Thirteenth International Conference on Automated Deduction in Geometry), Online, September 15-17, 2021. https://www.risc.jku.at/conferences/adg2021