

```

(*The Volume of the 3-sphere with constant curvature 1*)
VolOfS3=Pi*(2*Pi-Sin[2*Pi]);

(*These functions give the relationships of illustrated in Figure 2.1*)
ThetaFN[r1_,r2_] := ArcTan[Sqrt[3]*(Tan[r2]-Tan[r1])/(Tan[r1]+Tan[r2])];

xFN[r1_,r2_] := ArcTan[Tan[r1]*(Sqrt[3]/2*Cos[ThetaFN[r1,r2]]+1/2*
Sin[ThetaFN[r1,r2]])];

r3FN[r1_,r2_] := ArcCot[Cot[r1]-Cot[r2]];

Phi1FN[r1_,r2_] := ArcSin[Sin[xFN[r1,r2]]/Sin[r1]];

Psi1FN[r1_,r2_] := Module[{Phi1},Phi1=Phi1FN[r1,r2];If[(Tan[r2]-Tan[r1])/(
Tan[r1]+Tan[r2])-1/3<0,Pi-Phi1,Phi1]]

Phi2FN[r1_,r2_] := ArcSin[Sin[xFN[r1,r2]]/Sin[r2]];

Phi3FN[r1_,r2_] := ArcSin[Sin[xFN[r1,r2]]/Sin[r3FN[r1,r2]]];

(*Volume of a spherical cap in S3 of radius r and subtended by an angle
of 2 Phi *)
VolCap[r_,Phi_] := If[r\[Equal]Pi/2,(*
then*)0,(*else*)Pi(r-ArcTan[Cos[Phi]*Tan[r]]-Cos[r]*Sin[r]+Cos[
Phi]*Cos[r]*Sin[r])];

(*Area of a spherical cap in S3 of radius r and subtended by an angle
of 2 Phi *)
AreaCap[r_,Phi_] := 2Pi*(Sin[r]^2)*(1+Cos[Phi]);

(*Volume of a sphere of radius r in S3*)
VolSphere[r_] := Pi*(2*r-Sin[2*r]);

(*Area of a sphere of radius r in S3*)
AreaSphereGivenRadius[r_] := 4*Pi*(Sin[r])^2

(*Standard double bubbles are composed of two regions one enclosing the a
smaller volume and a larger volume. The exterior cap of the
smaller-volume-region has a radius of r1. The
exterior cap of the larger-volume-region has radius r2.*)
(*This function gives the volume of the smaller region of a standard double bubble
determined by radii r1, r2*)
VolOneSDB[r1_,r2_] := Module[{Psi1,r3,Phi3},Psi1=Psi1FN[r1,r2];
r3=r3FN[r1,r2];

```

```

Phi3=Phi3FN[r1,r2];
(*return*)VolCap[r1,Psi1]+VolCap[r3,Phi3]]

(*This function gives the volume of the larger region of a standard double bubble
determined by radii r1, r2*)
VolTwoSDB[r1_,r2_] :=Module[{Phi2,r3,Phi3},Phi2=Phi2FN[r1,r2];
r3=r3FN[r1,r2];
Phi3=Phi3FN[r1,r2];
(*return*)VolCap[r2,Pi-Phi2]-VolCap[r3,Phi3]]

(*When both regions of the double bubble are the same it becomes advantageous
to consider the one parameter function, that give the volume of one region given a
radius r.*)
EqualVolGivenRadius[r_] :=Pi/2*(
2r-Sin[2r])*(1+(Sqrt[2]*Cos[r])/(Sqrt[7+Cos[2r]]))+Pi(ArcTan[Sqrt[2]Sin[r]\
/(Sqrt[7+Cos[2r]])]-(Sqrt[2]*r*cos[r])/(Sqrt[7+Cos[2r]]));

(*Returns the sum of the areas of the three spherical caps that make up the standard
double bubble corresponding to the two given radii*)
AreaSDBGivenRadii[r1_,r2_] :=Module[{Psi1,Phi2,Phi3,r3},Psi1=Psi1FN[r1,r2];
Phi2=Phi2FN[r1,r2];
Phi3=Phi3FN[r1,r2];
r3=r3FN[r1,r2];
(*return*)AreaCap[r1,Pi-Psi1]+AreaCap[r2,Phi2]+AreaCap[r3,Pi-Phi3]]

(*Returns a radius of a sphere in S3 has less area than the sphere of volume v.
If v <= Vol0fS3/2, then this function returns a radius of a sphere with volume in
the interval [v-error, v]. If v > Vol0fS3/2, then
this function returns a radius of a sphere with volume in the interval [v, v+error] *)
RadiusSphere[v_, error_] :=Module[{minRadius, maxRadius, currentRadius,
counter},

counter =0;
minRadius=0;
maxRadius = Pi;
currentRadius = Pi/2;

If[v < Vol0fS3/2,

(*This while loop checks to see if the volume is in the interval [v-error, v] if not
then it checks to see if it the volume is too high or low. The While loop
improves the estimates minRadius and maxRadius. While the loop is running
minRadius corresponds to sphere of volume less than v-error and maxRadius
corresponds to a sphere of volume bigger than v.
Finally, the sphere of radius currentRadius corresponds to a sphere of volume
in [v-error, v] and the loop stops.*)

```

```

While[VolSphere[currentRadius] > v || (VolSphere[currentRadius] < v \
-error),

    If[VolSphere[currentRadius] > v,

        maxRadius= currentRadius;
        currentRadius = (currentRadius + minRadius)/2;

    ];

If[VolSphere[currentRadius] < v-error,

    minRadius= currentRadius;
    currentRadius = (currentRadius + maxRadius)/2;

];

counter++;

],

(*else if v > volofs3/2*)
(*This while loop checks to see if the volume is in the interval [v, v+error] if not
then it checks to see if it the volume is two high or low. The While loop
improves the estimate minRadius and maxRadius. While the loop is running
minRadius corresponds to sphere of volume less than v and maxRadius
corresponds to a sphere of volume bigger than v+error. Finally, the sphere of
radius currentRadius corresponds to a sphere of volume in [v, v+error]
and the loop stops.*)

While[VolSphere[currentRadius] < v || (VolSphere[currentRadius] > v \
+error),

    If[VolSphere[currentRadius] < v,

        minRadius= currentRadius;
        currentRadius = (currentRadius + maxRadius)/2;

    ];

If[VolSphere[currentRadius] > v + error,

    maxRadius= currentRadius;
    currentRadius = (currentRadius + minRadius)/2;

```

```

];

counter++;

];

];

(*Print["It took ", counter, "steps."];
Print["Radius is ", N[currentRadius]];*)
currentRadius
];

(*Gives an overestimate for the sphere of volume v*)
AreaSphereGivenVolume[v_, error_]:=AreaSphereGivenRadius[RadiusSphere[v, \
error]];

(* This function returns a pair of radii corresponding to a standard double bubble
with volumes bigger than v,w by making very small adjustments to the radii of the
exterior caps. This function will only work for v<= w < u. *)

RadiiSDB[v_, w_, vError_, wError_, vChangingFactor_,
wChangingFactor_]:=Module[{VRadius, WRadius, madeChangeToV, \
madeChangeToW, adjustedVError,
adjustedWError, vAdjustment, wAdjustment, volOne, volTwo, counter},

VRadius = Pi/4;
WRadius = Pi/4;
counter=0;

vAdjustment = vChangingFactor;
wAdjustment =wChangingFactor;

(*These variables make sure the v, w we approximate are in the region
we where A(v,w) increases in v and w*)
adjustedVError = vError;
adjustedWError = wError;

(*assurance that we are below line w =u and above line v=w*)
If[v+2w >= VolOfS3 || v >= w,

Print["ERROR RadiiSDB: v and w are not in increasingRegion!"];
Print["v+ 2w = " ,N[v+2w]];
Print["Vol of S3 = ", N[VolOfS3]];
Print["v = " ,v];

```

```

Print["w =",w];
,

While[v+adjustedVError + 2(w+adjustedWError) > VolOfS3 || \
v+adjustedVError > w+adjustedWError,

    If[v+adjustedVError > VolOfS3 - 2*w,

        adjustedVError= adjstedVError/2;

    ];

    If[2(w+adjustedWError) > VolOfS3 - (v + adjustedVError),

        adjustedWError= adjustedWError/2;
        adjustedVError= adjustedVError/Sqrt[2];

    ];

    If[v+adjustedVError > w+adjustedWError,

        adjustedVError = adjustedVError/2;

    ];

];

While[(VolOneSDB[VRadius, WRadius] < v || (VolOneSDB[VRadius,
WRadius]> v+
adjustedVError)||VolTwoSDB[VRadius,
WRadius] < w|| (VolTwoSDB[VRadius, WRadius]> \
w+adjustedWError)),

    (*These two variable prevent infinite while loop from occuring*)
    madeChangeToV = False;
    madeChangeToW = False;

    If[VolOneSDB[VRadius, WRadius]< v,

        VRadius = VRadius*vAdjustment;
        madeChangeToV= !madeChangeToV;

    ];

    (*This set of if statements adjusts one or both of the radii. If it can
    not change one of the radii then it refines the amount that the radii
    are adjusted by.*)

```

```

If[VolOneSDB[VRadius, WRadius]> v+adjustedVError,

    VRadius = VRadius/vAdjustment;
    madeChangeToV= !madeChangeToV;

];

If[VolTwoSDB[VRadius, WRadius]< w,

    WRadius = WRadius*wAdjustment;
    madeChangeToW= !madeChangeToW;

];

If[VolTwoSDB[VRadius, WRadius]> w+adjustedWError,

    WRadius = WRadius/wAdjustment;
    madeChangeToW= !madeChangeToW;

];

If[!madeChangeToV,

    vAdjustment=(vAdjustment+1)/2;

];

If[!madeChangeToW,

    wAdjustment=(wAdjustment+1)/2;

];

(*Print["Volume V at ", N[VolOneSDB[VRadius, WRadius]]];
Print["Volume W at ", N[VolTwoSDB[VRadius, WRadius]]];

Print["While Loop test
    is ", (VolOneSDB[VRadius, WRadius] < v || (VolOneSDB[VRadius, \
WRadius]> v+adjustedVError)||VolTwoSDB[VRadius, WRadius] <
        w || (VolTwoSDB[VRadius, WRadius]>
w+adjustedWError))];*)

];

```

```

];

(*Print["It took this many steps: ", counter];*)

{VRadius, WRadius}

];

```

(\*By symmetry the standard double bubble enclosing volumes (v,w,u) has the same area of the standard double bubble enclosing volumes (u,w,v). So, when w=u we can get an overestimate for the area of the standard double bubble enclosing volumes (v,w,u) by getting an overestimate for the area of the standard double bubble enclosing volumes (u,w,u). This function returns a pair of radii that correspond to a double bubble where the larger volume is equal to the exterior and both are bigger than or equal to w\*)

```

RadiiWhenWEqualsU[w_,
error_] := Module[{radius, minRadius, maxRadius, counter, adjustedError},
  minRadius = 0;
  maxRadius = 3Pi/2;
  radius = (minRadius + maxRadius/2);
  adjustedError = error;
  counter = 0;

```

```

If[3w < VolOfS3,

```

```

  Print["W is too small."],

```

```

  While[3(w - adjustedError) < VolOfS3,

```

```

    adjustedError = adjustedError/2;

```

```

  ];

```

```

(*Print["Target Range = (", N[w - error], ",", N[w], ")"]];*)

```

```

While[EqualVolGivenRadius[radius] > w ||
EqualVolGivenRadius[radius] < w - adjustedError,

```

```

  (*counter++;
  Print["counter = ", counter];*)

```

```

If[EqualVolGivenRadius[radius] > w,

```

```

  maxRadius = radius;
  radius = (minRadius + radius)/2;

```

```

];

If[EqualVolGivenRadius[radius]<w-adjustedError,

  minRadius =radius;
  radius = (maxRadius+radius)/2;

];

(*Print["Vol = ",N[EqualVolGivenRadius[radius]]];*)

];
(*Print["It took this many steps: ", counter];*)

];
{radius, radius}
];

```

```

(*This piece of code should be used when v=w*)
RadiiWhenVEqualsW[v_, error_] :=Module[{radius,
minRadius, maxRadius, counter, adjustedError},
  minRadius =0;
  maxRadius=Pi;
  radius = (minRadius+maxRadius/2);
  adjustedError=error;
  counter=0;

  (*Print["radius= ", radius];*)

  If[3v>= VolOfS3,

    Print["V is too big."],

    While[3(v+adjustedError)> VolOfS3,

      adjustedError = adjustedError/2;

    ];

    (*Print["Target Range = (", N[v-error],",", N[v],")" ];*)

    While[EqualVolGivenRadius[
radius]<v|| EqualVolGivenRadius[radius]>v+adjustedError,

```

```

(*counter++;
  Print["counter = ", counter];*)

If [EqualVolGivenRadius [radius]>v+adjustedError,

  maxRadius =radius;
  radius = (minRadius+radius)/2;

  ];

If [EqualVolGivenRadius [radius]<v,

  minRadius =radius;
  radius = (maxRadius+radius)/2;

  ];

(*Print["Vol = ",N[EqualVolGivenRadius[radius]]];*)

];

(*Print["It took this many steps: ", counter];*)

];

{radius, radius}

];

```

(\*This function returns an overestimate for the area of a standard double bubble of volumes v, w.\*)

```

A[v_,w_, vError_,
  wError_, vChangingFactor_, \
wChangingFactor_] :=Module[{radii,u,r1,r2,area},

  u=VolOfS3-v-w;

  (*when all three volumes are equal*)
  If[v==w && w==u,

    area = 6*Pi;

```

```

,
(*one the line v=w*)
If[v==w,

    radii = RadiiWhenVEqualsW[v, vError];
    area = AreaSDBGivenRadii[radii[[1]], radii[[2]]];

,
(*on the line w=u*)
If[w==u,

    radii = RadiiWhenWEqualsU[w, wError];
    area = AreaSDBGivenRadii[radii[[1]], radii[[2]]];

,
(*If all volumes are distinct*)

    radii = RadiiSDB[v, w, vError, wError, vChangingFactor,
    wChangingFactor];
    area = AreaSDBGivenRadii[radii[[1]], radii[[2]]];

];

];

];

area
];

```

(\*This function is described in Section 4 of the paper. It shows that either the Hutchings function is on a rectangular domain or it breaks up the rectangle into 4 smaller rectangles and checks the code again. The function completes and returns a 1 if the function is positive on the rectangular domain and 0 if the approximated function is negative on any part of the domain\*)

```

ProofFunctionRectangle[lhs_,rhs_,p1_,p3_,left11_,right11_,left33_,
    right33_,depth_] := Module[{decision,x1,y1,x2,y2,x3,
    y3,left12,right12,left13,right13,
    left21,right21,left22,right22,left23,right23,left31,
    right31,left32,right32,d1,d2,d3,d4},depth=1;

```

```

x1=p1[[1]];
y1=p1[[2]];

```

```

x3=p3[[1]];
y3=p3[[2]];

```

```

x2=(x1+x3)/2;
y2=(y1+y3)/2;

left12=lhs[x1,y2];right12=rhs[x1,y2];
left13=lhs[x1,y3];right13=rhs[x1,y3];
left21=lhs[x2,y1];right21=rhs[x2,y1];
left22=lhs[x2,y2];right22=rhs[x2,y2];
left23=lhs[x2,y3];right23=rhs[x2,y3];
left31=lhs[x3,y1];right31=rhs[x3,y1];
left32=lhs[x3,y2];right32=rhs[x3,y2];

(*if x1 > y3 then balancing
covers the region and we don't need to check it.*)
If[x1 > y3,

    decision =1,

    (*otherwise we do*)
    If[left11>right11&&left33>right33,

        (*then*)

        If[Min[left11,left13,left31,left33]>right33,(*then*)Print["Points \
",p1," and ",p3," -- Direct hit!"];
        decision=1,

        (*else*)Print["Points ",p1," and ",
p3," -- Splitting into four."];

        If[ProofFunctionRectangle[lhs,rhs,{x1,y1},{x2,y2},left11,right11,
left22,right22,d1]\[Equal]1&&
ProofFunctionRectangle[lhs,rhs,{x2,y1},{x3,y2},left21,right21,\
left32,right32,d2]\[Equal]1&&
ProofFunctionRectangle[
lhs,rhs,{x1,y2},{x2,y3},left12,right12,left23,right23,
d3]\[Equal]1&&ProofFunctionRectangle[lhs,rhs,{x2,y2},{
x3,y3},left22,right22,left33,right33,d4]\[Equal]1,

            (*then*)
            Print["Points ",p1," and
",p3," -- Hit after checking four! Depth: ",Max[depth,
d1+1,d2+1,d3+1,d4+1]];
            decision=1,

            (*else*)Print["Proof function failed"];

```

```

        decision=0;
        (*endif*);
        depth=Max[depth,d1+1,d2+1,d3+1,d4+1];
        (*endif*)),Print["Oh no for ",p1," and ",p3,"!"];
        Print[N[left11]];
        Print[N[right11]];
        Print[N[left33]];
        Print[N[right33]];
        decision=0;
    ];

    (*endif*)
];
(*return*)decision
];

```

(\*This function is described in Section 4 of the paper. It shows that either the Hutchings function is on a triangular domain or it breaks up the triangle into 2 smaller triangles and a rectangle and checks the code again. The function completes when it has broken the domain into small enough domains to see that the function is postive and returns a 1 or if the approximated function is negative on any part of the domain the function returns a 0.\*)

```

ProofFunctionTriangle[lhs_,rhs_,x1_,y1_,x3_,y3_,left11_,right11_,left13_,\
right13_,left31_,right31_,depth_]:=Module[{decision,x2,y2,left12,right12,\
left21,right21,left22,right22,d1,d2,d3},depth=1;

```

```

    Print["starting up the process"];

```

```

    x2=(x1+x3)/2;

```

```

    y2=(y1+y3)/2;

```

```

    Print["whoa now!"];

```

```

    If[left11>right11&&left13>right13&&left31>right31,(*then*)

```

```

        If[Min[left11,
left13,left31]>right31,(*then*)Print["Points ",{x1,y1}," ",{x1,
y3}," and ",{x3,y1}," -- Direct triangle hit!"];
        decision=1,

```

```

        (*else*)Print["Points ",{x1,y1}," ",{x1,y3}," and ",{x3,y1}," --
        Splitting into three."];

```

```

        left12=lhs[x1,y2];right12=rhs[x1,y2];

```

```

        left21=lhs[x2,y1];right21=rhs[x2,y1];

```

```

        left22=lhs[x2,y2];right22=rhs[x2,y2];

```

```

        If[ProofFunctionRectangle[lhs,rhs,{x1,y1},{x2,y2},left11,right11,\

```

```

left22,right22,d1]\[Equal]1&&ProofFunctionTriangle[lhs,rhs,x1,y2,x2,y3,left12,
    right12,left13,right13,left22,right22,d2]\[Equal]1&&\
ProofFunctionTriangle[lhs,rhs,x2,y1,x3,y2,left21,
    right21,left22,right22,left31,right31,d3]\[Equal]1,(*then*)
    Print["Points ",{x1,y1}," ",{x1,y3}," and ",{x3,y1}," --
    Hit after checking three! Depth: ",Max[depth,d1+1,d2+1,d3+1]];
decision=1,(*else*)Print[
    "Points ",{x1,y1}," ",{x1,y3}," and ",{x3,y1}," -- Failed
    after checking three!"];
decision=0;
(*endif*);
depth=Max[depth,d1+1,d2+1,d3+1];
(*endif*),(*else*)Print["Oh no for " {x1,y1},"
    ",{x1,y3}," and ",{x3,y1},"!"];
Print[N[left11]];
Print[N[right11]];
Print[N[left13]];
Print[N[right13]];
Print[N[left31]];
Print[N[right31]];
decision=0;
(*endif*);
(*return*)decision]

```