

H3 Code

```
(*error control in computation the default value of maching*)
WorkingPrecision->MachinePrecision;
COMPUTERERROR= 2^-24;
ACCURACY = 25;

LargestSingCurvature = 16.8;

AreaSphere[k_Real]:=SetAccuracy[(4*Pi)/(-1+k^2)-COMPUTERERROR, ACCURACY];

VolSphere[k_Real]:=SetAccuracy[Pi*(-2*
  ArcCoth[k]+Sinh[2*ArcCoth[k]]),ACCURACY];

ASC1close[k1_Real,k2_Real]:=SetAccuracy[(2*Pi*(1+Sqrt[-((k1^2*(-1+Cos[(1/6)*(\
Pi-6*ArcTan[(Sqrt[3]*(k1-k2))/(
  k1+k2)]])^2)))/(
  k1^2-Cos[(1/6)*(
  Pi-6*ArcTan[(
  Sqrt[3]*(k1-
  k2))/(k1+k2)]])^2)))]/(-1+k1^2), \
ACCURACY+1];

ASC1far[k1_Real,
  k2_Real]:=SetAccuracy[-((2*Pi*(-1+Sqrt[-((k1^2*(-1+Cos[(1/6)*(Pi-6*\
ArcTan[(Sqrt[3]*(k1-k2))/(
  k1+k2)]])^2)))/(k1^2-Cos[(
  1/6)*(Pi-6*ArcTan[(
  Sqrt[3]*(
  k1-k2))/(k1+k2)]])^2)))]/(-1+
  k1^2)), ACCURACY+1];

ASC1[k1_Real,k2_Real]:=SetAccuracy[If[(k1-k2)/(k1+k2)-1/3<0,ASC1close[
  k1,k2],ASC1far[k1,k2]], ACCURACY+1];

ASC2[k1_Real,k2_Real]:=SetAccuracy[(2*Pi*(1+Sqrt[(k1^2-k2^2*Cos[(1/6)*(Pi-6*
  ArcTan[(Sqrt[3]*(k1-k2))/(k1+
  k2)]])^2)/(
  k1^2-Cos[(1/
  6)*(Pi-6*ArcTan[(Sqrt[3]*(k1-k2))/(k1+
  k2)]])^2)))]/(-1+k2^2), ACCURACY+1];

ASC3[k1_Real,k2_Real]:=SetAccuracy[(2*Pi*(1-Sqrt[(k1^2-(k1-k2)^2*Cos[(
  1/6)*(Pi-6*
```

```

ArcTan[(Sqrt[3]*(
    k1-k2))/(k1+
    k2))]^2)/(k1^2-
    Cos[(1/6)*(
    Pi-6*ArcTan[(
    Sqrt[3]*(k1-
    k2))/(k1+k2))]^2)))/(-1+(k1-k2)^2), \
ACCURACY+1];

AreaDb1Bubble[k1_Real,k2_Real]:=SetAccuracy[ASC1[k1,k2]+ASC2[k1,
    k2]+ASC3[k1,k2]+3*COMPUTERERROR, ACCURACY];

VolCap1close[k1_Real,k2_Real]:=-((1/(-1+k1^2))*(Pi*((-1+k1^2)*
    ArcCoth[k1]+(-1+k1^2)*ArcTanh[Sqrt[-((k1^2*(-1+Cos[(1/6)*(Pi-6*ArcTan[(\
Sqrt[3]*(k1-k2))/(k1+k2)]]^2)))/(k1^2-Cos[(1/6)*(Pi-6*ArcTan[(Sqrt[3]*(k1-k2))\
/(k1+k2)]]^2)))/k1]-k1*(1+Sqrt[-((
    k1^2*(-1+Cos[(1/6)*(Pi-6*ArcTan[(Sqrt[\
3*(k1-k2))/(k1+k2)]]^2)))/(k1^2-Cos[(1/6)*(Pi-6*ArcTan[(
    Sqrt[3]*(k1-k2))/(k1+k2)]]^2)))]))));

VolCap1far[k1_Real,k2_Real]:=Pi*(-ArcCoth[k1]+(1/(-1+k1^2))*(k1+(-1+k1^2)*\
ArcTanh[Sqrt[-((k1^2*(-1+Cos[(1/6)*(Pi-6*ArcTan[(Sqrt[
    3*(k1-k2))/(
    k1+k2)]]^2)))/(k1^2-Cos[(1/6)*(Pi-6*ArcTan[(Sqrt[
    3*(k1-
    k2))/(k1+
    k2)]]^2)))]/
    k1]-k1*
    Sqrt[-((k1^2*(-1+Cos[(1/6)*(Pi-6*\
ArcTan[(Sqrt[3]*(k1-k2))/(k1+k2)]]^2)))/(k1^2-Cos[(
    1/6)*(Pi-6*ArcTan[(Sqrt[3]*(k1-k2))/(k1+
    k2)]]^2)))]));

VolCap1[k1_Real,k2_Real]:=If[(k1-k2)/(k1+k2)-1/3<0,
    VolCap1close[k1,k2],VolCap1far[k1,k2]];

VolCap2[k1_Real,k2_Real]:=Pi*(-(k2/(1-k2^2))-ArcCoth[k2]-
    ArcTanh[Sqrt[(k1^2-k2^2*Cos[(1/6)*(Pi-6*ArcTan[(Sqrt[3]*(
    k1-k2))/(k1+k2)]]^2)/(k1^2-Cos[(
    1/6)*(Pi-6*
    ArcTan[(Sqrt[3]*(
    k1-k2))/(k1+k2)]]^2)]/k2)-(k2*Sqrt[(k1^2-\
k2^2*Cos[(1/6)*(Pi-6*ArcTan[(Sqrt[3]*(k1-k2))/(k1+k2)]]^2)/(k1^2-
    Cos[(1/6)*(Pi-6*ArcTan[(Sqrt[
    3*(k1-k2))/(k1+k2)]]^2)))]/(1-k2^2));

VolCap3[k1_Real,
    k2_Real]:=Re[Pi*(ArcTanh[Sqrt[(k1^2-(k1-k2)^2*Cos[(1/6)*(Pi-6*ArcTan[(\

```

```

Sqrt[3]*(k1-k2))/(k1+k2))]^2)/(k1^2-
      Cos[(1/6)*(Pi-6*ArcTan[(Sqrt[
      3*(k1-k2))/(k1+k2))]^2)]/(k1-k2)]+(1/(-\
1+k1^2-2*k1*k2+k2^2))*(k1-k2-(-1+k1^2-2*
      k1*k2+k2^2)*
      ArcCoth[
      k1-k2]+(-k1+
      k2)*Sqrt[(k1^2-(k1-k2)^2*Cos[(1/6)*(Pi-6*ArcTan[\
(Sqrt[3]*(k1-k2))/(k1+k2))]^2)/(k1^2-Cos[(1/6)*(Pi-6*ArcTan[(
      Sqrt[3]*(k1-k2))/(k1+k2))]^2)))]);

```

```

VolBub1[k1_Real,k2_Real]:=If[k1 < 1 || k2 < 1,
  Print["VolBub1 Error : k1 or k2 too small"];
  Print["k1 = ", k1, " k2 = ", k2 ];
  ,If[k1 \[Equal] k2,VolCap1[k1,k2]
  ,
  VolCap1[k1,k2]+VolCap3[k1,k2]
  ]
];

```

```

VolBub2[k1_Real,k2_Real]:=If[k1\[Equal]k2, VolCap2[k1,k2]
,
VolCap2[k1,k2]-VolCap3[k1,k2]
];

```

```

VolBub1A[k1_Real, k2_Real]:=If[k1>= k2,
  VolBub1[k1, k2]
,
(*Print["VolBub1A curvatures switched"];*)
VolBub2[k2, k1]
];

```

```

VolBub2A[k1_Real, k2_Real]:=If[k1>= k2,
  VolBub2[k1, k2]
,
(*Print["VolBub2A curvatures switched to ", VolBub1[k1, k2]]);*)
VolBub1[k2, k1]
];

```

```

VolBubV[k1_Real, k2_Real] := SetAccuracy[VolBub1A[k1, k2], ACCURACY];

```

```

VolBubW[k1_Real, k2_Real] := SetAccuracy[VolBub2A[k1, k2], 15];

```

(*gives an underapproximation for the curvature of a sphere in H3*)

```
CurvatureEstimatorSingle[v_Real] := If[v > .0001,
```

```
  If[ v < .008,
```

```
    8.109,
```

```
  If[ v<.02,
```

```
    6.,
```

```
  If[
```

```
    (*.02 v < 1*)
```

```
    v < 1,
```

```
    1.84871,
```

```
  If[
```

```
    (*1 v < 10*)
```

```
    v<10,
```

```
    1.19394,
```

```
  If[
```

```
    (*20 v < 30 *)
```

```
    v < 30,
```

```
    1.08108,
```

```
  If[
```

```
    (*30 v < 50*)
```

```
    v<50,
```

```
    1.05231,
```

```
  If[
```

```
    (*50 v < 60*)
```

```
    v< 60,
```

```
    1.04456,
```

```
  If[
```

```
    (*60 <= v < 70*)
```

```
    v< 70,
```

```
    1.03883,
```

```
  If[
```

```
    (*70 v < 80*)
```

```
    v< 80,
```

```
    1.03437,
```



```

While[VolSphere[curvature] > v - COMPUTERERROR || VolSphere[curvature] \
< v-error ,

    curvature = (bigK+smallK)/2;
    counter = counter +1;

    (*this tightens the interval that curvatures can be in*)
    If[VolSphere[curvature] > v - COMPUTERERROR,

        smallK= curvature;

        ,If[VolSphere[curvature] < v - error,

            bigK = curvature;

            ];
        ];
    ];
    (*Print["It took ", counter, "steps." ];*)
    curvature
];

```

(*this function returns curvature pair that is at least one box size over \ then returns the array with the new curvature pair and the changes for each \ curvature though this seems to do the same thing as CurvaturesfromVolDouble[] \ this doesn't limit the size of the change*)

```

MakeChangeOneBoxSizeOver[k1_Real, k2_Real, volume1_Real, BoxSize_Real, \
change1_Real] := Module[{minVolChange, maxVolChange, adjCurvature1, \
adjCurvature2, adjChange1, finalArray, counter, slope, closestOverShot, \
closestUnderShot, lastCurvatureGuess},

```

```

    minVolChange = .8*BoxSize;
    maxVolChange = 1.7*BoxSize;

```

```

    adjCurvature1 = change1*k1;
    adjCurvature2 = k2;
    adjChange1 = change1;
    counter=0;

```

```

    closestOverShot= k2;
    closestUnderShot = k1;

```

```

    slope = (VolBub1A[adjCurvature1, adjCurvature2] - \
volume1)/(adjCurvature1 - k1);

```

```

    adjCurvature1 = adjCurvature1 + BoxSize/slope;

```

```

(Print["curvatures, k1 = ",adjCurvature1, " k2 is ", k2];*)

While[ (VolBub1A[adjCurvature1,adjCurvature2]- volume1< minVolChange || \
VolBub1A[adjCurvature1,adjCurvature2]- volume1 > maxVolChange) && \
adjCurvature1 > 1 && counter < 10,

  If[VolBub1A[adjCurvature1,adjCurvature2]- volume1< minVolChange,

    (*Print["hi test 1 is ", VolBub1A[
adjCurvature1,adjCurvature2]- volume1< minVolChange];*)

    lastCurvatureGuess = adjCurvature1;

    adjCurvature1 = (adjCurvature1 + closestOverShot)/2;

    closestOverShot = lastCurvatureGuess;
  ];

  If[VolBub1A[adjCurvature1,adjCurvature2]- volume1> maxVolChange,

    (*Print["hi test 2 is ", VolBub1A[adjCurvature1,adjCurvature2]-
volume1> maxVolChange];*)

    lastCurvatureGuess = adjCurvature1;

    adjCurvature1 = (adjCurvature1 + closestUnderShot)/2;

    closestUnderShot = lastCurvatureGuess;
  ];

  (*Print["trial", counter, " vol is \
",VolBub[adjCurvature1,adjCurvature2]];*)

  counter= counter+1;
  (*Print["counter = ", counter];
  Print["vol1 =",
  VolBub1A[adjCurvature1,adjCurvature2] ];
  Print["change = \
", adjChange1];
  Print["curvature ", adjCurvature1 ];
  *)

];

If[adjCurvature1 <= 1,

```

```

Print["MakeChangeOneBoxSizeOver error curvature too small"];
];
(*
Print["vol1 =", VolBub1[adjCurvature1,adjCurvature2] ];
Print["change = ", adjChange1];
Print["curvature ", adjCurvature1];
*)
adjChange1 = adjCurvature1/k1;

{adjCurvature1, adjChange1}
];

(*this function returns curvature pair that is at least one box size over \
then returns the array with the new curvature pair and the changes for each \
curvature though this seems to do the same thing as CurvaturesfromVolDouble[] \
this doesn't limit the size of the change*)
MakeChangeOneBoxSizeUp[k1_Real, k2_Real, volume2_Real, BoxSize_Real, \
change2_Real] := Module[{minVolChange, maxVolChange,adjCurvature1, \
adjCurvature2, adjChange1, counter, slope, lastCurvatureGuess, \
closestUnderShot, closestOverShot},

minVolChange = 1.*BoxSize;
maxVolChange = 2.*BoxSize;

adjCurvature1 = k1;
adjCurvature2 = k2*change2;
adjChange2 = change2;
closestUnderShot=k2;
closestOverShot = 1.;
(*Might want to change this!*)

counter=0;

(*Print["hi"];
Print["test =", (VolBub2[adjCurvature1,adjCurvature2]- volume2< \
minVolChange || VolBub2[adjCurvature1,
adjCurvature2]- volume2 > maxVolChange) &&
adjCurvature2 > 1 && counter < 10];
*)

slope = (
VolBub2A[adjCurvature1, adjCurvature2] - volume2)/(adjCurvature2 - k2);

adjCurvature2 = adjCurvature2 + BoxSize/slope;

While[ (VolBub2A[adjCurvature1,

```

```

adjCurvature2]- volume2< minVolChange || \
VolBub2A[adjCurvature1,adjCurvature2]-
    volume2 > maxVolChange) && adjCurvature2 > 1  && counter < 10,

counter= counter+1;
(*Print["counter = ", counter];
  Print["vol2 =", VolBub2[adjCurvature1,adjCurvature2] ];
  Print["change = ", adjChange2];
  Print["curvature ", adjCurvature1 ];
  *)

If[VolBub2A[adjCurvature1,adjCurvature2]- volume1< minVolChange,

  lastCurvatureGuess = adjCurvature2;

  adjCurvature2 = (adjCurvature2 +  closestOverShot)/2;

  closestOverShot = lastCurvatureGuess;
  ];

If[VolBub2A[adjCurvature1,adjCurvature2]- volume1> maxVolChange,

  lastCurvatureGuess = adjCurvature1;

  adjCurvature2 = (adjCurvature2 +  closestUnderShot)/2;

  closestUnderShot = lastCurvatureGuess;
  ];

];

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

If[adjCurvature2 <= 1,

  Print["MakeChangeOneBoxSizeUp error curvature too small"];
  ];
  (*
  Print["vol2 =", VolBub2[adjCurvature1,adjCurvature2] ];
  Print["change = ", adjChange2];
  Print["curvature ", adjCurvature2];
  *)
adjChange2 = adjCurvature2/k2;

{adjCurvature2, adjChange2}
];

```

```
(*This function tells whether the point is in given rectangle with lower left \
corner xValueBox, yValueBox and width boxWidth and heigh boxHeight*)
IsPointinBox[xValuePoint_Real, yValuePoint_Real, xValueBox_Real, \
yValueBox_Real, boxWidth_Real, boxHeight_Real] :=
  (xValuePoint> xValueBox+ COMPUTERERROR && xValuePoint < (xValueBox + \
boxWidth) && yValuePoint > yValueBox + COMPUTERERROR &&
  yValuePoint < (yValueBox + boxHeight));
```

```
(*This function tells whether the point has a x value and a value bigger than \
a given x, and y value *)
IsPointToUpperRight[xValuePoint_Real, yValuePoint_Real, xValueBox_Real, \
yValueBox_Real] :=
  (xValuePoint> xValueBox) && yValuePoint > yValueBox ;
```

```
(*This function is supposed manipulate a given curvature pair for volumes \
vol1 and vol2 and get a curvature pair corresponding to volumes in inside a \
box with v1, v2 as its lower left corner*)
(*right now this method needs to dyanically choose its changing size to avoid \
infinite loops*)
(*Returns a two element list of curvatures*)
(*v1, v2 should correspond to the point in the lower right of the rectangle \
containing VolBub1A[k1, k2], VolBub2A[k1, k2]
the curvatures k1, k2 should be a point in the middle of the rectangle*)
```

```
NextCurvaturePairinArray[ k1_Real, k2_Real, v1_Real, v2_Real, change1_Real, \
change2_Real, scalingFactor1_Real, scalingFactor2_Real] \
:=Module[{curvatureComponent11, curvatureComponent12, curvatureComponent21, \
curvatureComponent22, startingVolume1, startingVolume2, volumeComponent11, \
volumeComponent12, volumeComponent21, volumeComponent22, alpha, beta, \
volumeOneTarget, volumeTwoTarget, curvaturePair, volumeChangeMatrix,
  startingVoltoTargetMatrix, alphaBetaMatrix, counter, \
oneBoxSizeOverArray, oneBoxSizeUpArray},
```

```
(*adjustment arrays*)
oneBoxSizeOverArray =
  MakeChangeOneBoxSizeOver[k1, k2,
  VolBub1A[k1,k2], change1, scalingFactor1];

  oneBoxSizeUpArray = MakeChangeOneBoxSizeUp[k1, k2, VolBub2A[k1, k2], \
change2, scalingFactor2];
```

```
(*Declaration of initial curvaturevectors*)
```

```

curvatureComponent11 = oneBoxSizeOverArray[[1]];

curvatureComponent12 = k2;
curvatureComponent21 = k1;
curvatureComponent22 = oneBoxSizeUpArray[[1]];
counter=0;

(*Print["c11 = ", curvatureComponent11];
  Print["c12 = ", curvatureComponent12];
  Print["c21 = ", curvatureComponent21];
  Print["c22 = ", curvatureComponent22];*)

(*declaration of initial volumevectors*)
startingVolume1 = VolBub1A[k1, k2];
startingVolume2 = VolBub2A[k1, k2];
volumeComponent11 = VolBub1A[
  curvatureComponent11, curvatureComponent12];
volumeComponent12 = VolBub2A[curvatureComponent11, \
curvatureComponent12];
volumeComponent21 = VolBub1A[
  curvatureComponent21, curvatureComponent22];
volumeComponent22 = VolBub2A[curvatureComponent21, \
curvatureComponent22];

(*Print["v11 = ", volumeComponent11];
  Print["v12 = ", volumeComponent12];
  Print["v21 = ", volumeComponent21];
  Print["v22 = ", volumeComponent22];*)

(*declaration of target Volumes*)
volumeOneTarget = v1 + .5*change1;
volumeTwoTarget = v2 + .5*change2;

(*Print["volumeOneTarget = ", volumeOneTarget];
  Print["volumeTwoTarget = ", volumeTwoTarget];
  *)

(*declaration of Matrices*)
startingVoltoTargetMatrix= {{volumeOneTarget -
  startingVolume1}, {volumeTwoTarget - startingVolume2}};

(*Print["first test =
  ",IsPointinBox[volumeComponent11, volumeComponent12, v1+change1, \
v2, change1, change2]];
  *)

```

```

If[IsPointinBox[volumeComponent11, volumeComponent12, v1, v2, change1,
  change2],

(*Print["v11 = ", volumeComponent11];
  Print["v12 = ", volumeComponent12];
  Print["change1 = ", change1];
  Print["change2 = ", change2];
  *)

curvaturePair = {curvatureComponent11, curvatureComponent12},

(*Print["2nd test = ", IsPointinBox[volumeComponent21, \
volumeComponent22, v1+change1, v2, change1, change2]]];

  Print["v21 = ", volumeComponent21];
  Print["v22 = ", volumeComponent22];
  Print["change1 = ", change1];
  Print["change2 = ", change2];
  *)

If[IsPointinBox[volumeComponent21, volumeComponent22, v1, v2, change1,
  change2],

(*Print["case II"];
  Print["v21 = ", volumeComponent21];
  Print["v22 = ", volumeComponent22];
  Print["change1 = ", change1];
  Print["change2 = ", change2];*)

curvaturePair = {curvatureComponent21, curvatureComponent22},

(*Print["while loop test", \
!IsPointinBox[volumeComponent11, volumeComponent12, v1+change1, v2, change1,
  change2]]];
  *)

  volumeChangeMatrix = {{volumeComponent11 - startingVolume1, \
volumeComponent12 - startingVolume2}, {volumeComponent21 -
  startingVolume1, volumeComponent22 - startingVolume2}};

(*Print["volumeChangeMatrix = ", volumeChangeMatrix//MatrixForm];
  *)

alphaBetaMatrix = Inverse[volumeChangeMatrix] .
  startingVoltoTargetMatrix;
(*Print["alphaBetaMatrix = ", alphaBetaMatrix//MatrixForm];
  *)

```

```

alpha = alphaBetaMatrix[[1,1]];
beta = alphaBetaMatrix[[2,1]];

curvatureComponent11 = (curvatureComponent11-k1)*alpha + \
(curvatureComponent21-k1)*beta +k1;
curvatureComponent12= (curvatureComponent12-k2)*
alpha + (curvatureComponent22-k2)*beta + k2;

(*Print["inside while loop"];
Print["c11 = ", curvatureComponent11];
Print["c12 = ", curvatureComponent12];
Print["alpha = ", alpha];
Print["beta = ", beta];*)

volumeComponent11 = VolBub1A[
curvatureComponent11, curvatureComponent12];
volumeComponent12 = VolBub2A[
curvatureComponent11, curvatureComponent12];

(*Print["v11 = ", volumeComponent11];
Print["v12 = ", volumeComponent12];*)

];

curvaturePair = {curvatureComponent11, curvatureComponent12};
(*Print["after loop"];
Print["v11 = ", volumeComponent11];
Print["v12 = ", volumeComponent12];
Print["change1 = ", change1];
Print["change2 = ", change2];

Print["2nd
test = ",IsPointinBox[volumeComponent11, \
volumeComponent12, v1+change1, v2, change1, change2]];*)

];

(*If[!
IsPointinBox[VolBub1[curvatureComponent11, curvatureComponent12], \
VolBub2[curvatureComponent11, curvatureComponent12], v1+change1, v2, \
change1, change2],

curvaturePair = {-1, -1};
];
*)
curvaturePair

```

```

];

(*This returns a lower bound on the positive part of the Hutchings function \
for a given volume pair*)
LowerBoundOnPosHutchingsFunction[vol1_Real, vol2_Real, error1_Real, \
error2_Real]:=

2*AreaSphere[CurvaturefromVolSingle[vol1/2, error1]]+
AreaSphere[CurvaturefromVolSingle[vol2, error2]]+
AreaSphere[CurvaturefromVolSingle[vol1+vol2, error2]];

(*This returns an array of areas is indexed by its volume pair *)
ArrayBuilderForSingAreasandCurvatures[vMin_Real, vMax_Real, boxSize_Real, \
error_Real]:=Module[
{AreaArray, volumeArray, n, g},

g[i_Integer]= AreaSphere[CurvaturefromVolSingle[vMin+ i*boxSize, \
error]];

n[i_Integer]=VolSphere[CurvaturefromVolSingle[vMin+ i*boxSize, error]];

AreaArray= Array[g, Ceiling[(vMax-vMin)/boxSize], 0];
volumeArray=Array[n, Ceiling[(vMax-vMin)/boxSize], 0];

{volumeArray,AreaArray}
];

(*This returns an array of areas is indexed by its volume pair *)
ArrayBuilderForSingAreasandVolumes[vMin_Real, vMax_Real, boxSize_Real, \
error_Real]:=Module[
{AreaArray, volumeArray, n, g},

g[i_Integer]= AreaSphere[CurvaturefromVolSingle[vMin+i*boxSize,
error]];
n[i_Integer]=vMin+i*boxSize;

AreaArray= Array[g, Ceiling[(vMax-vMin)/boxSize], 0];
volumeArray=Array[n, Ceiling[(vMax-vMin)/boxSize], 0];

{volumeArray,AreaArray}
];

```

```

(*This returns an array of areas is indexed by its volume pair *)
ArrayBuilderForSingAreas[vMin_Real, vMax_Real, \
boxSize_Real, error_Real]:=Module[
  {AreaArray, g},

  g[i_Integer]= AreaSphere[CurvaturefromVolSingle[vMin+i*boxSize, \
error]];

  AreaArray= Array[g, Ceiling[(vMax-vMin)/boxSize], 0];

  AreaArray
];

```

```

(*This code fills an array with curvature pairs. It moves on to the next \
region after verifying that 2 the area of the double bubble enclosing volumes \
v,w is less than the concave part of the Hutchings function for v,w*)
ArrayFillingProof[vMin_Real, wMin_Real,wMax_Real,rectangleHeight_Real, \
rectangleWidth_Real,startingCurvature1_Real, startingCurvature2_Real, \
adjustmentMainVol_Real,adjustmentSecondVol_Real] :=Module[{smallerVolume, \
largerVolume,nextRowStartingCurvatures, curvaturePair, failSafe,
  nextRowStartingPosition,counter, insideCounter, failingVolumeV, \
singAreasArray, singAreasArrayStart},

  failSafe=True;
  counter=0;
  insideCounter =1;

  singAreasArrayStart=(vMin-rectangleWidth)/2;
  singAreasArray= ArrayBuilderForSingAreas[(vMin-rectangleWidth)/2, \
2*(wMax+rectangleWidth), rectangleWidth/2, rectangleWidth/2];
  Print["Array completed, evaluating Hutchings Function"];
  (*Print["singAreasArray =", singAreasArray];*)

  (*Print["hi"];
  Print[failSafe];*)

  (*nextRowStartingPosition keeps tract of the x position that \
starts a new row
  initially it is set to 0 and then set to correct position*)
  nextRowStartingPosition = 0.0;
  nextRowStartingCurvatures = curvaturePair;

  curvaturePair = {startingCurvature1, startingCurvature2};

```

```

smallerVolume = vMin;
largerVolume=wMin+rectangleHeight;

(*Print["first test", largerVolume wMax +rectangleHeight && \
failSafe];*)

While[largerVolume wMax +rectangleHeight && failSafe,

  (*While the v1 is less than v2*)
  While[smallerVolume largerVolume +rectangleWidth && failSafe ,

    failSafe = IsPointinBox[VolBubV[curvaturePair[[
1]], curvaturePair[[2]]],
    VolBubW[curvaturePair[[1]],curvaturePair[[2]]], smallerVolume, \
largerVolume, 2*rectangleWidth, 2*rectangleHeight];

    If[!failSafe,

      Print["( ",
        VolBubV[curvaturePair[[1]], curvaturePair[[2]]], ", ",
        VolBubW[curvaturePair[[1]],
          curvaturePair[[2]]], ") is not in the
          box defined by (", smallerVolume,", ", largerVolume,) \
and (", rectangleWidth, ", ", rectangleHeight, ")" ];

    ];
  (*marker*)
  (*this sets up moving one row up*)
  If[ nextRowStartingPosition \[Equal]0 && .85 * largerVolume \
smallerVolume+rectangleWidth,
    nextRowStartingPosition = smallerVolume;
    nextRowStartingCurvatures = curvaturePair;

  ];

  (*Print["hi! 11"];*)
  (*If[insideCounter <= 5,
    Print["target volume pair = {" , smallerVolume, ",", largerVolume,
    "}"];
    Print["actual volume

          pair = {" , \
VolBubV[curvaturePair[[1]],curvaturePair[[2]] ], ",", \
VolBubW[curvaturePair[[1]],curvaturePair[[2]] ], "}"];
    Print["curvatures are ", curvaturePair];

```

```

        insideCounter++;
        Print["failSafe ", failSafe];

    ];*)

    (*if there is a problem with the curvature function or the \
    hutchings function is negative then the program should fail*)

    (*Print["v/2 from array =", \
    singAreasArray[[Floor[((smallerVolume-rectangleWidth)/2-singAreasArrayStart)/(\
    rectangleWidth/2)+1]]]];
        Print["index =", \
    Floor[((smallerVolume-rectangleWidth)/2-singAreasArrayStart)/(rectangleWidth/\
    2)+1]];

        Print["v/2 not from array = " ,
                AreaSphere[CurvaturefromVolSingle[(
                smallerVolume-rectangleWidth)/2, \
rectangleWidth/2]]];

    Print["w from array =",
        singAreasArray[[Floor[
            2+(largerVolume-rectangleWidth-singAreasArrayStart)/(\
rectangleWidth/2)]]]];

        Print["index =", \
    Floor[2+(largerVolume-rectangleWidth-singAreasArrayStart)/(rectangleWidth/2)]]\
;

        Print["w not from array = " \
,AreaSphere[CurvaturefromVolSingle[largerVolume-
        rectangleWidth, rectangleWidth/2]]];

    Print["v+w from array =", \
    singAreasArray[[Floor[1+(smallerVolume+largerVolume-rectangleWidth-
        singAreasArrayStart)/(rectangleWidth/2)]]]];

        Print["index =", \
    Floor[1+(-rectangleWidth+smallerVolume+largerVolume-singAreasArrayStart)/(\
rectangleWidth/2)]];

    Print[
        "v+w not from array = " \

```

```

,AreaSphere[CurvaturefromVolSingle[smallerVolume+largerVolume-rectangleWidth, \
rectangleWidth/2]]];

Print["v + w =", smallerVolume+largerVolume-rectangleWidth];
Print["v = ", smallerVolume];
Print["w = ", largerVolume-rectangleWidth];

*)

If[curvaturePair[[1]] < 1 \
||2*singAreasArray[[Floor[((smallerVolume-rectangleWidth)/2-\
singAreasArrayStart)/(rectangleWidth/2)+1]]]+singAreasArray[[
Floor[2+(largerVolume-rectangleWidth-\
singAreasArrayStart)/(rectangleWidth/
2)]]]+singAreasArray[[Floor[1+(smallerVolume+\
largerVolume-rectangleWidth-singAreasArrayStart)/(rectangleWidth/2)]]]+ - \
2*AreaDblBubble[curvaturePair[[1]], curvaturePair[[2]]] < 0,

Print["curvaturePair =", curvaturePair];
Print["hutchings function = ", \
2*singAreasArray[[Floor[((smallerVolume-rectangleWidth)/2-
singAreasArrayStart)/(rectangleWidth/2)+1]]]+singAreasArray[[\
Floor[2+(largerVolume-rectangleWidth-singAreasArrayStart)/(rectangleWidth/2)]]\
]+singAreasArray[[Floor[1+(smallerVolume+largerVolume-rectangleWidth-\
singAreasArrayStart)/(rectangleWidth/2)]]] - 2*AreaDblBubble[
curvaturePair[[1]], curvaturePair[[2]]]];
failSafe = False;
failVolumeV =smallerVolume;

Print["upper
bound on dbl bubble", 2*AreaDblBubble [curvaturePair[[1]], \
curvaturePair[[2]]]];
Print["lower bound on concave",singAreasArray[[
Floor[((smallerVolume-rectangleWidth)/2-\
singAreasArrayStart)/(rectangleWidth/2)+1]]]+singAreasArray[[Floor[2+(\
largerVolume-
rectangleWidth-singAreasArrayStart)/(rectangleWidth/2)]]]+\
singAreasArray[[Floor[1+(smallerVolume+largerVolume-rectangleWidth-
singAreasArrayStart)/(rectangleWidth/2)]]]];

];

(*Print["inside while loop"];*)
curvaturePair = NextCurvaturePairinArray[curvaturePair[[1]], \
curvaturePair[[2]], smallerVolume+rectangleWidth, largerVolume,
rectangleWidth, rectangleHeight, adjustmentMainVol, \

```

```

adjustmentSecondVol];

    (*Print["upper bound on dbl bubble", 2*AreaDblBubble \
[curvaturePair[[1]], curvaturePair[[2]]]];
    Print["lower bound on \
concave", LowerBoundOnPosHutchingFunction[smallerVolume, largerVolume, \
rectangleWidth, rectangleHeight]];
    *)

    smallerVolume = smallerVolume+ rectangleWidth;

];

If[failSafe,
    (*Print["smallerVolume", smallerVolume];*)
    smallerVolume = nextRowStartingPosition;

    nextRowStartingPosition =0.0;
    counter++;
    insideCounter= 1;

    If[counter \[Equal]100,
        Print["w at ", largerVolume];
        counter=0;
    ];

    (*Print["SETTING the curvature pair"];
    Print["Volumes before {", \
VolBubV[nextRowStartingCurvatures[[1]],nextRowStartingCurvatures[[2]]], " , \
",VolBubW[nextRowStartingCurvatures[[1]],
    nextRowStartingCurvatures[[2]]], "]" ];

    Print["target box { (" , smallerVolume ",", \
smallerVolume+rectangleHeight , ") X (
        ", (largerVolume+rectangleHeight), ", ", \
(largerVolume+2*rectangleHeight)" )}"];
    *)
    largerVolume = largerVolume+ rectangleHeight;

    While[!IsPointinBox[VolBubV[curvaturePair[[1]],
        curvaturePair[[2]]], VolBubW[curvaturePair[[1]],curvaturePair[[
        2]]], smallerVolume, largerVolume, 2*rectangleWidth, \
2*rectangleHeight],

```

```

        curvaturePair = NextCurvaturePairinArray[curvaturePair[[1]], \
curvaturePair[[2]], smallerVolume, largerVolume+rectangleHeight, \
rectangleWidth, rectangleHeight, adjustmentMainVol, adjustmentSecondVol];
    ];
    (*Print["VOL1 =",VolBubV[curvaturePair[[1]], curvaturePair[[2]]] ];
    Print["VOL2 =",VolBubW[curvaturePair[[1]], curvaturePair[[2]]] ];
    *)

    failSafe=IsPointinBox[VolBubV[curvaturePair[[1]],curvaturePair[[2]]]\
, VolBubW[curvaturePair[[1]],curvaturePair[[2]]], smallerVolume, largerVolume, \
2*rectangleWidth, 2*rectangleHeight]&&smallerVolume/largerVolume.85;

    If[!failSafe,

        Print["error at end ( ",VolBubV[
            curvaturePair[[1]], curvaturePair[[2]]], ",
            ", VolBubW[curvaturePair[[1]],curvaturePair[[2]]], ")
            is not in the box defined
            by (", smallerVolume," ", ", largerVolume,")
            and (", rectangleWidth, ", ", ", rectangleHeight, ")" ];

    ];

    (*Print["test ",IsPointinBox[VolBubV[curvaturePair[[1]],
curvaturePair[[
2]]], VolBubW[curvaturePair[[1]],curvaturePair[[2]]], smallerVolume,
    largerVolume, rectangleWidth, rectangleHeight]];*)
];

];

(*Print["while loop test ", smallerVolume    largerVolume && failSafe, \
failSafe];
Print["first part of test", smallerVolume    largerVolume];
*)

If[failSafe,
    Print["done"],

    Print["failed at (",smallerVolume, ", ", ", largerVolume, ")"];
    Print["ratio =",smallerVolume/largerVolume];
];

```

```
Print["failSafe = ", failSafe];  
];
```