Preamble

```
SetDirectory[NotebookDirectory[]]
```

```
/Users/ecbrown/src/QTAIM.wl/QTAIM
```

Parallel Kernels

```
LaunchKernels[];
```

```
(*CloseKernels[]*)
```

```
Needs["QTAIM`"];
ParallelNeeds["QTAIM`"]
```

```
(* Simple command to test parallel initialization: *)
(*ParallelEvaluate[$QTAIMContoursPositive]*)
```

MathLink

You may compile a MathLink executable to load Fortran-formatted WFN files.

```
link = Install[$HomeDirectory <> "/src/QTAIM.wl/QTAIM/QTAIM/qtaim"];
```

```
(*Uninstall[link];*)
(*ParallelEvaluate[Uninstall[link]];*)
```

Wavefunction

Import Existing Wavefunction from Web Server:

```
w = Import["https://ericcbrown.com/QTAIM/wdx/water-cas1010-augccpvtz-opt.wdx"];
```

Alternative ways to import wavefunctions:

From a WFX on a HTTPS Server (Water)

```
w = ReadWavefunctionFromWFX[
    "https://ericcbrown.com/QTAIM/wfx/water-hf-ccpvdz-opt.wfx"];
```

From a WFX on a HTTPS Server (Pyridine)

From a WDX on a HTTPS Server (Insulin)

```
w = Import["https://ericcbrown.com/QTAIM/wdx/insulin_hf.wdx"];
```

From PySCF

From a WFN File

Export Examples

```
(*Export["../resource/water-cas1010-augccpvtz-opt.wdx",w]*)

(*Export["../resource/water-cas1010-augccpvtz-pyscf.wdx",w]*)

(*Export["../resource/water-cas1010-ccpvdz-pyscf.wdx",w]*)

(*Export["../resource/insulin_hf.wdx",w]*)
```

Electron Density and its Derivatives

The electron density, its gradient, and its Hessian convenience functions are defined so that:

```
rho[w, {0., 0., 0.}]
```

```
10.7958
```

```
g[w, {0., 0., 0.}]
```

$$\{-7.64557\times10^{-20},\ -3.95407\times10^{-17},\ 154.216\}$$

```
H[w, {0., 0., 0.}]
```

$$\{\{-699.31,\ 6.58808\times10^{-17},\ 1.16299\times10^{-16}\},\ \{6.58808\times10^{-17},\ -712.381,\ -1.30566\times10^{-16}\},\ \{1.16299\times10^{-16},\ -1.30566\times10^{-16},\ 2374.99\}\}$$

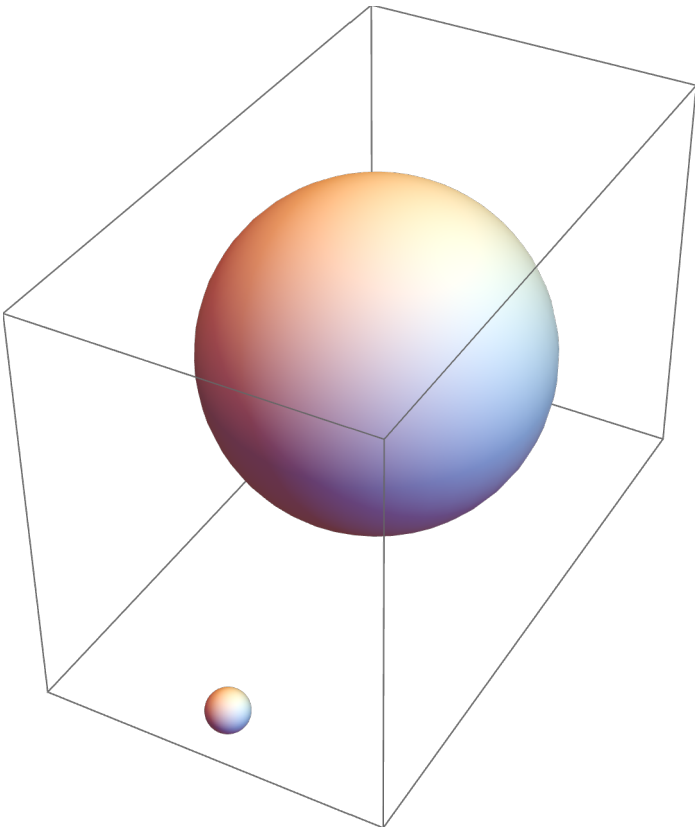
Nuclear Critical Points

```
ncps = LocateNuclearCriticalPoints[w] // Chop;
```

```
ncps // TableForm
```

```
0    0           0.218693
0    1.32401     -0.809514
0    -1.32401    -0.809514
```

```
Graphics3D[
 Table[
  Sphere[ncps[[i]], 0.1 * ((w["AtomicNumbers"])[[i]])]
  , {i, 1, Length[ncps]}]
]
```



Bounding Box

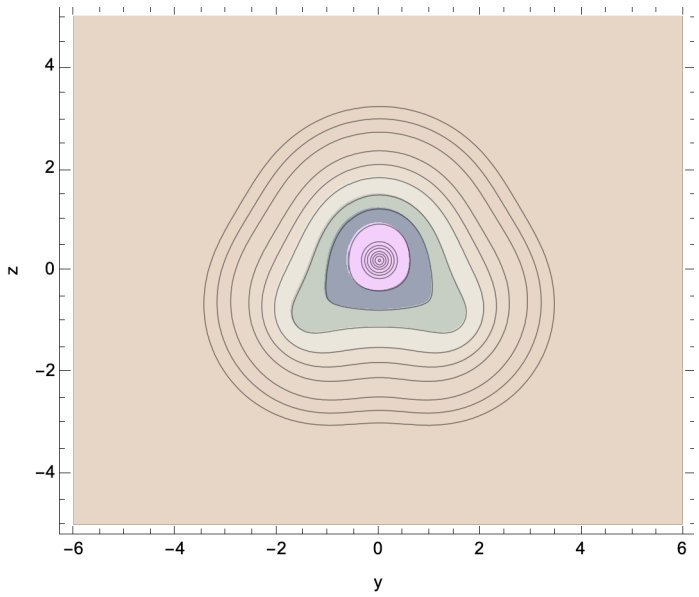
```
bbpad = 4;
bbxmin = Floor[Min[ncps[[All, 1]]] - bbpad;
bbxmax = Ceiling[Max[ncps[[All, 1]]] + bbpad;
bbymin = Floor[Min[ncps[[All, 2]]] - bbpad;
bbymax = Ceiling[Max[ncps[[All, 2]]] + bbpad;
bbzmin = Floor[Min[ncps[[All, 3]]] - bbpad;
bbzmax = Ceiling[Max[ncps[[All, 3]]] + bbpad;
bb = {{bbxmin, bbxmax}, {bbymin, bbymax}, {bbzmin, bbzmax}};
```

The bounding box:

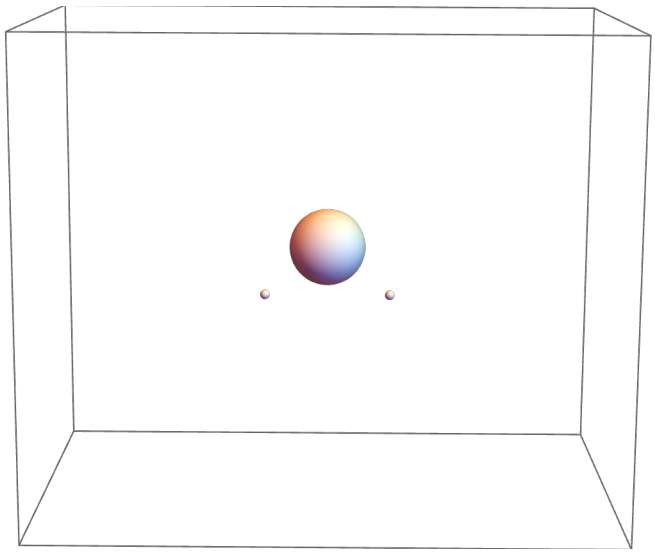
```
N@bb
```

```
{{-4., 4.}, {-6., 6.}, {-5., 5.}}
```

```
rhoPlot = ContourPlot[
  rho[w, {0., y, z}],
  {y, bbymin, bbymax},
  {z, bbzmin, bbzmax},
  FrameLabel → {"y", "z"},
  Contours → $QTAIMContoursPositive,
  PlotRange → All
  , PlotPoints → 50
  , ColorFunction → "PearlColors"
  , ColorFunctionScaling → False
  , AspectRatio → Automatic]
```



```
Graphics3D[
 Table[
  Sphere[ncps[[i]], 0.1 * ((w["AtomicNumbers"])[[i]])]
  , {i, 1, Length[ncps]}]
 , PlotRange → bb]
```



Subdivision

Small Molecules

It is beneficial to make a grid that has the nuclear critical points on the corners. This only works for small molecules!

```
xrange = Join[{-Infinity}, Sort[DeleteDuplicates[
      Chop[ncps[[All, 1]], (EuclideanDistance[#1, #2] < 10^-6) &]], {Infinity}];
yrange = Join[{-Infinity}, Sort[DeleteDuplicates[
      Chop[ncps[[All, 2]], (EuclideanDistance[#1, #2] < 10^-6) &]], {Infinity}];
zrange = Join[{-Infinity}, Sort[DeleteDuplicates[
      Chop[ncps[[All, 3]], (EuclideanDistance[#1, #2] < 10^-6) &]], {Infinity}];
```

Make sure the number of subdivisions is reasonable:

```
Length[
 Flatten[
  Table[1,
   {xmin, 1, Length[xrange] - 1},
   {ymin, 1, Length[yrange] - 1},
   {zmin, 1, Length[zrange] - 1}
  ]
  , 2]
]
```

```
24
```

```
q = AbsoluteTiming[
  ParallelSum[
   NIntegrate[
    If[rho[w, {x, y, z}] > Min[$QTAIMContoursPositive],
     rho[w, {x, y, z}],
     0
    ],
    {x, xrange[[xmin]], xrange[[xmin + 1]]},
    {y, yrange[[ymin]], yrange[[ymin + 1]]},
    {z, zrange[[zmin]], zrange[[zmin + 1]]},
    Method → {"LocalAdaptive", "SymbolicProcessing" → 0}
    , AccuracyGoal → 4, PrecisionGoal → Infinity],
   {xmin, 1, Length[xrange] - 1},
   {ymin, 1, Length[yrange] - 1},
   {zmin, 1, Length[zrange] - 1}
   , Method → "FinestGrained"]
 ]
```

```
{3.09926, 9.89666}
```

Large Molecules

For large molecules, it can be beneficial to make a grid that is on the order of the number of processors one has.

```
xrange = Subdivide[bbxmin, bbxmax, 3];
yrange = Subdivide[bbymin, bbymax, 3];
zrange = Subdivide[bbzmin, bbzmax, 3];
```

```
Length[
 Flatten[
  Table[1,
   {xmin, 1, Length[xrange] - 1},
   {ymin, 1, Length[yrange] - 1},
   {zmin, 1, Length[zrange] - 1}
  ]
  , 2]
]
```

```
27
```

```
q = AbsoluteTiming[
  NIntegrate[
   rho[w, {x, y, z}],
   {x, bbxmin, bbxmax},
   {y, bbymin, bbymax},
   {z, bbzmin, bbzmax},
   Method → {"LocalAdaptive", "SymbolicProcessing" → 0}
   , AccuracyGoal → 4, PrecisionGoal → Infinity
  ]
 ]
```

```
{0.566439, 9.99946}
```

Parallel Example

Here, an even number of subdivisions is helpful so that the nuclear critical points are on a boundary. Then, NIntegrate will give them special treatment as it does for all boundaries.

```
xrange = Subdivide[bbxmin, bbxmax, 2];
yrange = Subdivide[bbymin, bbymax, 2];
zrange = Subdivide[bbzmin, bbzmax, 2];

q = AbsoluteTiming[
  ParallelSum[
   NIntegrate[
    rho[w, {x, y, z}],
    {x, xrange[[ix]], xrange[[ix + 1]]},
    {y, xrange[[iy]], xrange[[iy + 1]]},
    {z, xrange[[iz]], xrange[[iz + 1]]},
    Method → {"LocalAdaptive", "SymbolicProcessing" → 0}
    , AccuracyGoal → 2, PrecisionGoal → Infinity
   ]
   , {ix, Length[xrange] - 1}
   , {iy, Length[yrange] - 1}
   , {iz, Length[zrange] - 1}
  ]
 ]
```

```
{0.118139, 9.98016}
```

```
{0.089777, 9.98016}
```

```
{0.070798, 9.98956}
```

```
{0.071745, 9.98956}
```

Molecular Graph

Evaluate the molecular graph, all critical points corresponding to nuclear, bond, ring, and cage critical points. This may be slower than heuristic methods, e.g. start between two NCPs but it is general, for use in cases where one may not even know where to begin. (It solves tetrahedrane, see last example.)

(It is normal to see CompiledFunction and other errors here. These will be quieted in a later version.)

```
AbsoluteTiming[
 cps = LocateCriticalPoints[
   {
    gx[w, {x, y, z}],
    gy[w, {x, y, z}],
    gz[w, {x, y, z}]
   },
   {x, bbxmin, bbxmax},
   {y, bbymin, bbymax},
   {z, bbzmin, bbzmax}
  ]
]
```

CompiledFunction: Numerical error encountered; proceeding with uncompiled evaluation.

Set: Lists {QTAIM`Private`cp000$110424, QTAIM`Private`cp100$110424} and {0, 0, 0, 0, 0} are not the same shape.

Part: Part specification QTAIM`Private`cp100$110424[[1]] is longer than depth of object.

Part: Part specification QTAIM`Private`cp000$110424[[1]] is longer than depth of object.

Part: Part specification QTAIM`Private`cp100$110424[[2]] is longer than depth of object.

General: Further output of Part::partd will be suppressed during this calculation.

CompiledFunction: Numerical error encountered; proceeding with uncompiled evaluation.

Set: Lists {QTAIM`Private`cp000$110453, QTAIM`Private`cp010$110453} and {0, 0, 0, 0, 0} are not the same shape.

CompiledFunction: Numerical error encountered; proceeding with uncompiled evaluation.

General: Further output of CompiledFunction::cfne will be suppressed during this calculation.

Set: Lists {QTAIM`Private`cp000$110462, QTAIM`Private`cp001$110462} and {0, 0, 0, 0, 0} are not the same shape.

General: Further output of Set::shape will be suppressed during this calculation.

FindRoot: The function value

{4. QTAIM`Private`cp000$110424[[1]] QTAIM`Private`cp100$110424[[1]] + 4. QTAIM`Private`cp000$110424[[2]] QTAIM`Private`cp100$110424[[2]] + ≪1≫ + 4. ≪26≫[[4]] QTAIM`Private`cp100$110424[[4]] + 4. QTAIM`Private`cp000$110424[[5]] QTAIM`Private`cp100$110424[[5]], ≪1≫, ≪1≫} is not a list of numbers with dimensions {3} at {x, y, z} = $\{5.85454\times10^{-17}, 1.74847\times10^{-8}, -15.9364\}$.

FindRoot: The function value

{4. QTAIM`Private`cp000$110477[[1]] QTAIM`Private`cp100$110477[[1]] + 4. QTAIM`Private`cp000$110477[[2]] QTAIM`Private`cp100$110477[[2]] + ≪1≫ + 4. ≪26≫[[4]] QTAIM`Private`cp100$110477[[4]] + 4. QTAIM`Private`cp000$110477[[5]] QTAIM`Private`cp100$110477[[5]], ≪1≫, ≪1≫} is not a list of numbers with dimensions {3} at {x, y, z} = $\{8.25864\times10^{-15}, 6.67763\times10^{-8}, 16.507\}$.

FindRoot: The function value

{4. QTAIM`Private`cp000$110482⟦1⟧ QTAIM`Private`cp100$110482⟦1⟧ + 4. QTAIM`Private`cp000$110482⟦2⟧ QTAIM`Private`cp100$110482⟦2⟧ + ≪1≫ + 4. ≪26≫⟦4⟧ QTAIM`Private`cp100$110482⟦4⟧ + 4. QTAIM`Private`cp000$110482⟦5⟧ QTAIM`Private`cp100$110482⟦5⟧, ≪1≫, ≪1≫} is not a list of numbers with dimensions {3} at {x, y, z} = $\{5.85454\times10^{-17}, 1.74847\times10^{-8}, -15.9364\}$.

General: Further output of FindRoot::nlnum will be suppressed during this calculation.

$$\{133.225,\ \{\{-1.64478\times10^{-16}, -1.32401, -0.809514\}, \{-1.44992\times10^{-16}, -1.14644, -0.677648\},\ \{7.34835\times10^{-20}, 1.32401, -0.809514\}, \{7.34835\times10^{-20}, 1.32401, -0.809514\},\ \{1.60903\times10^{-19}, 1.14644, -0.677648\}, \{1.60903\times10^{-19}, 1.14644, -0.677648\}\}\}$$

Characterize the rank and signature, and ellipticity of the critical points with these helper functions:

```
CriticalPointRank[{x_, y_, z_}] :=
  Total[Map[If[# == 0, 0, 1] &, Chop[Eigenvalues[H[w, {x, y, z}]]]]];
CriticalPointSignature[{x_, y_, z_}] := Total[
   Map[Which[# < 0, -1, # == 0, 0, # > 0, 1] &, Chop[Eigenvalues[H[w, {x, y, z}]]]]];
Ellipticity[{x_, y_, z_}] := Module[
   {evals = Sort[Chop[Eigenvalues[H[w, {x, y, z}], -2]]]}, (evals⟦1⟧ / evals⟦2⟧) - 1];
```

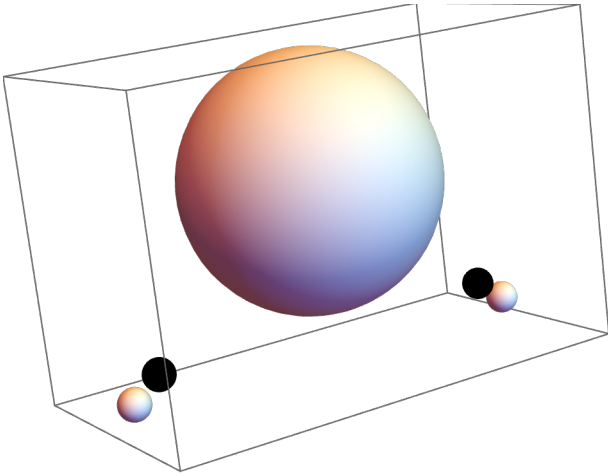
```
uniqueCps = DeleteDuplicates[Chop[cps], EuclideanDistance[#1, #2] < 1*^-8 &]
```

```
{{0, -1.32401, -0.809514}, {0, -1.14644, -0.677648},
 {0, 1.32401, -0.809514}, {0, 1.14644, -0.677648}}
```

```
cpsTable = Select[
  Transpose[
   {
    uniqueCps⟦All, 1⟧,
    uniqueCps⟦All, 2⟧,
    uniqueCps⟦All, 3⟧,
    Map[CriticalPointRank, uniqueCps],
    Map[CriticalPointSignature, uniqueCps],
    Map[Ellipticity, uniqueCps]
   }
  ]
  , #⟦5⟧ ≠ -3 &]
```

```
{{0, -1.14644, -0.677648, 3, -1, -2.39404}, {0, 1.14644, -0.677648, 3, -1, -2.39404}}
```

```
Graphics3D[
 Join[
  Table[
   {Black, Sphere[cpsTable[[i, 1 ;; 3]], 0.1]}
   , {i, 1, Length[cpsTable]}]
  ,
  Table[
   Sphere[ncps[[i]], 0.1 * ((w["AtomicNumbers"])[[i]])]
   , {i, 1, Length[ncps]}]
 ]
]
```



```
bcps = Select[
   Transpose[
    {
     uniqueCps[[All, 1]],
     uniqueCps[[All, 2]],
     uniqueCps[[All, 3]],
     Map[CriticalPointRank, uniqueCps],
     Map[CriticalPointSignature, uniqueCps]
    }
   ]
   , #[[5]] == -1 &];
```

```
rcps = Select[
   Transpose[
    {
     uniqueCps[[All, 1]],
     uniqueCps[[All, 2]],
     uniqueCps[[All, 3]],
     Map[CriticalPointRank, uniqueCps],
     Map[CriticalPointSignature, uniqueCps]
    }
   ]
   , #[[5]] == 1 &];

ccps = Select[
   Transpose[
    {
     uniqueCps[[All, 1]],
     uniqueCps[[All, 2]],
     uniqueCps[[All, 3]],
     Map[CriticalPointRank, uniqueCps],
     Map[CriticalPointSignature, uniqueCps]
    }
   ]
   , #[[5]] == 3 &];

bcps // TableForm
```

```
0    -1.14644    -0.677648    3    -1
0    1.14644     -0.677648    3    -1
```

```
rcps // TableForm
```

```
{}
```

```
ccps // TableForm
```

```
{}
```

Parallel Algorithm (WIP)

Unlike the case of integration, it is helpful to make an odd number of divisions, so that the planes of symmetry are not on a boundary. (Parallelization is probably not necessary or could be worse for small molecules)

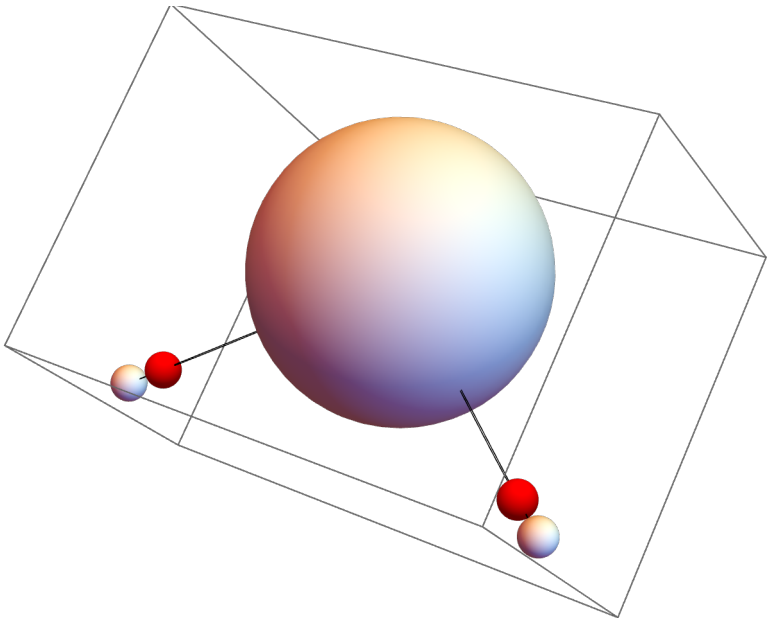
```
(*xrange=Subdivide[bbxmin,bbxmax,3];
yrange=Subdivide[bbymin,bbymax,3];
zrange=Subdivide[bbzmin,bbzmax,3];

AbsoluteTiming[
 cps=ParallelTable[
   LocateCriticalPoints[
    {
     gx[w,{x,y,z}],
     gy[w,{x,y,z}],
     gz[w,{x,y,z}]
    },
    {x,xrange[[ix]],xrange[[ix+1]]},
    {y,yrange[[iy]],yrange[[iy+1]]},
    {z,zrange[[iz]],zrange[[iz+1]]}
    (*,Method→{"Newton"},Jacobian:→H[w,{x,y,z}]*)
   ]
   ,{ix,Length[xrange]-1}
   ,{iy,Length[yrange]-1}
   ,{iz,Length[zrange]-1}
  ]
]*)
```

Bond Paths

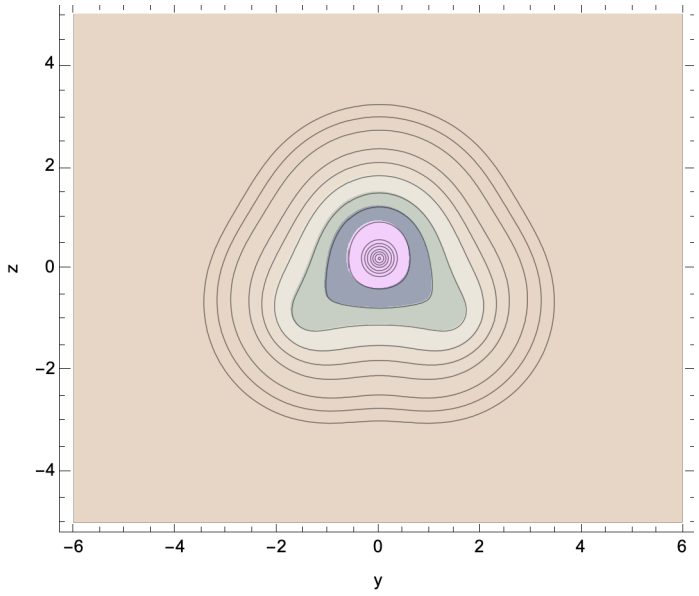
```
bondPaths = ParallelTable[
   BondPath[
    w, ncps, bcps[[bp, 1 ;; 3]]
   ]
   , {bp, 1, Length[bcps]}];
```

```
bondPathLines3D = ParallelTable[
    Line[
     Join[
       bondPaths[[
         bp, (*bp*)
         1, (*dir*)
         2, 1]],
       Reverse[bondPaths[[
          bp, (*bp*)
          2, (*dir*)
          2, 1]]
      ][[All, 2]]
    ]
    , {bp, 1, Length[bondPaths]}];
```

```
graph = Show[
  Graphics3D[
   bondPathLines3D
  ]
  ,
  Graphics3D[
   Join[
    Table[
     {Red, Sphere[bcps[[i, 1 ;; 3]], 0.1]}
     , {i, 1, Length[bcps]}]
    ,
    Table[
     {Green, Sphere[rcps[[i, 1 ;; 3]], 0.1]}
     , {i, 1, Length[rcps]}]
    ,
    Table[
     {Blue, Sphere[ccps[[i, 1 ;; 3]], 0.1]}
     , {i, 1, Length[ccps]}]
    ,
    Table[
     Sphere[ncps[[i]], 0.1 * ((w["AtomicNumbers"])[[i]])]
     , {i, 1, Length[ncps]}]
   ]
  ]
 ]
```

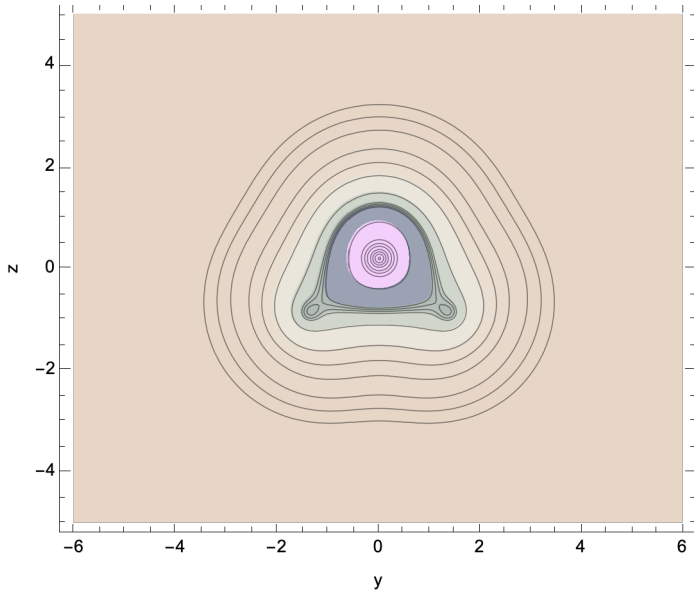


```
bondPathLines = Table[
    Line[
     Join[
         bondPaths[[
           bp, (*bp*)
           1, (*dir*)
           2, 1]],
         Reverse[bondPaths[[
            bp, (*bp*)
            2, (*dir*)
            2, 1]]]
        ][[All, 2]][[All, {2, 3}]]
    ]
    , {bp, 1, Length[bondPaths]}];

rhoPlot = ContourPlot[
   rho[w, {0., y, z}],
   {y, bbymin, bbymax},
   {z, bbzmin, bbzmax},
   FrameLabel → {"y", "z"},
   Contours → $QTAIMContoursPositive,
   PlotRange → All
   , PlotPoints → 50
   , ColorFunction → "PearlColors"
   , ColorFunctionScaling → False
   , AspectRatio → Automatic]
```

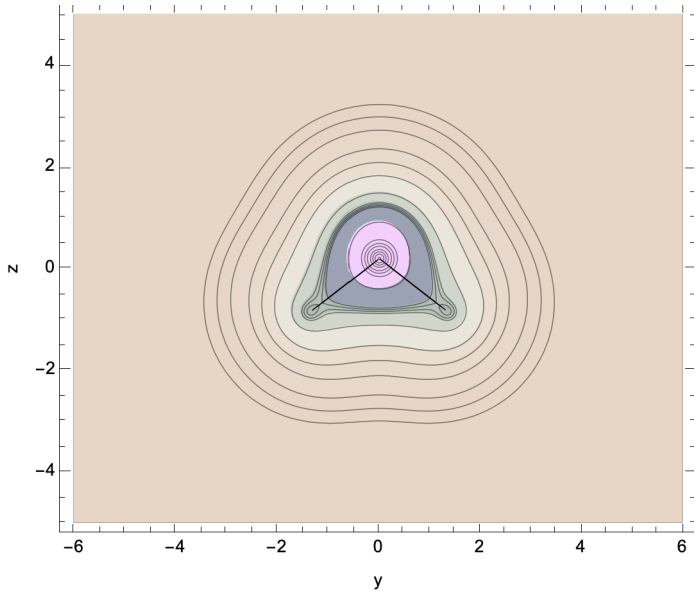


With some trial and error, additional contours illustrate BCPs:

```
rhoPlot = ContourPlot[
  rho[w, {0., y, z}],
  {y, bbymin, bbymax},
  {z, bbzmin, bbzmax},
  FrameLabel → {"y", "z"},
  Contours →
   DeleteDuplicates[Sort[Join[$QTAIMContoursPositive, {13 / 40, 14 / 40, 15 / 40}]]],
  PlotRange → All
  , PlotPoints → 50
  , ColorFunction → "PearlColors"
  , ColorFunctionScaling → False
  , AspectRatio → Automatic]
```

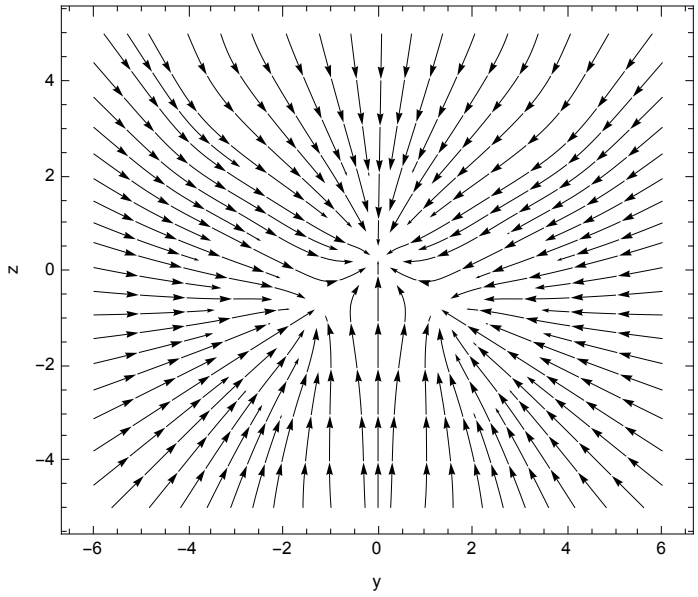


```
Show[
 rhoPlot,
 Graphics[bondPathLines]
]
```

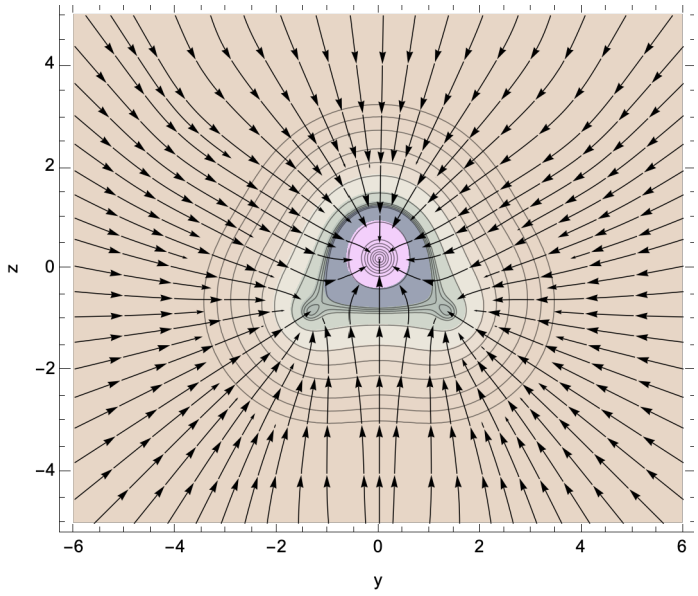


Gradient of the Electron Density

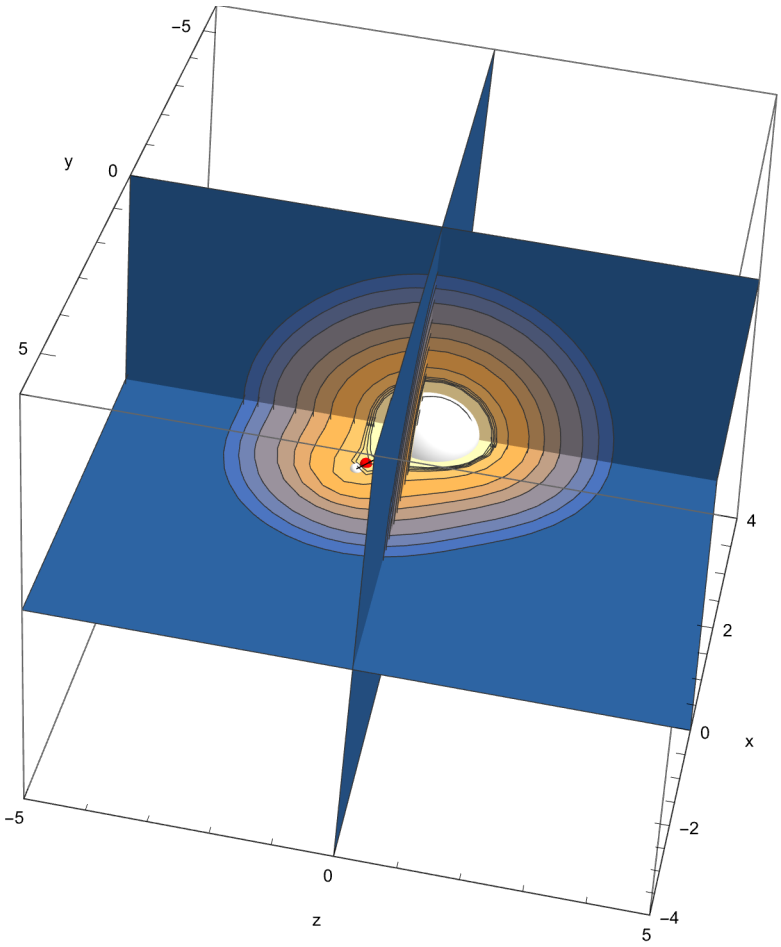
```
gyz[wfn_, {y_?NumericQ, z_?NumericQ}] := Rest[g[wfn, {0, y, z}]];

streamPlot = StreamPlot[
  gyz[w, {y, z}],
  {y, bbymin, bbymax},
  {z, bbzmin, bbzmax},
  FrameLabel → {"y", "z"}
  , StreamStyle → Black
  , StreamColorFunction → None
  , AspectRatio → Automatic]
```



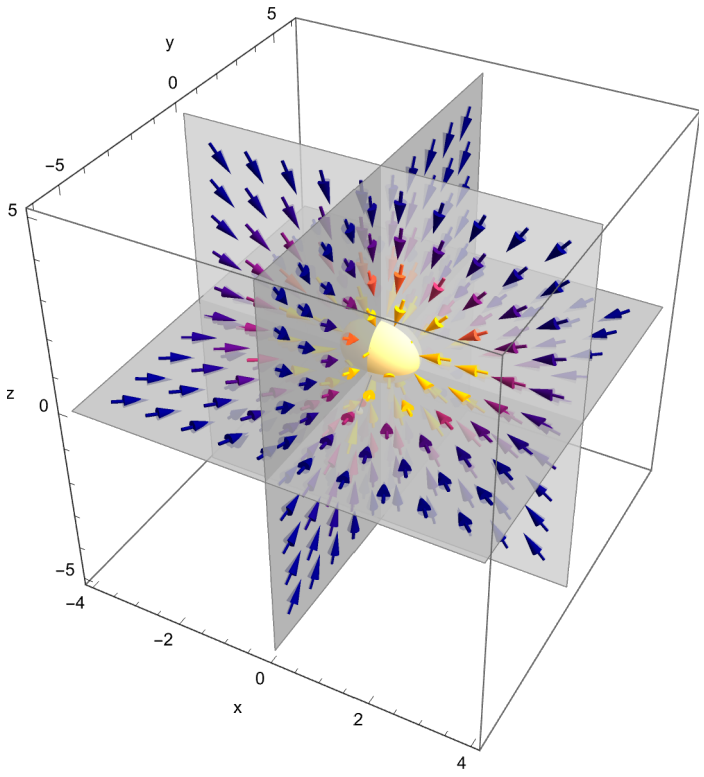
```
Show[
 rhoPlot,
 streamPlot
]
```



```
Show[
 SliceContourPlot3D[
  rho[w, {x, y, z}],
  "CenterPlanes",
  {x, bbxmin, bbxmax},
  {y, bbymin, bbymax},
  {z, bbzmin, bbzmax},
  AxesLabel → {"x", "y", "z"},
  Contours →
   DeleteDuplicates[Sort[Join[$QTAIMContoursPositive, {13 / 40, 14 / 40, 15 / 40}]]],
  PlotRange → {Full, Full, Full, All}
  , BoxRatios → Automatic
 ]
 ,
 graph
]
```



```
Show[
 SliceVectorPlot3D[
  g[w, {x, y, z}],
  "CenterPlanes",
  {x, bbxmin, bbxmax},
  {y, bbymin, bbymax},
  {z, bbzmin, bbzmax},
  AxesLabel → {"x", "y", "z"},
  PlotRange → All
  , BoxRatios → Automatic]
 ,
 graph
]
```



(Negative of the) Laplacian of the Electron Density

It is fairly easy to define custom fields. For example, the Laplacian, its gradient, and its Hessian can be expressed using the ElectronDensityDerivative convenience function.

```
laprho[wfn_, {x_?NumericQ, y_?NumericQ, z_?NumericQ}] := Total[
   First[
    ElectronDensityDerivative[wfn, {{x, y, z}},
     {{2, 0, 0}, {0, 2, 0}, {0, 0, 2}}
```

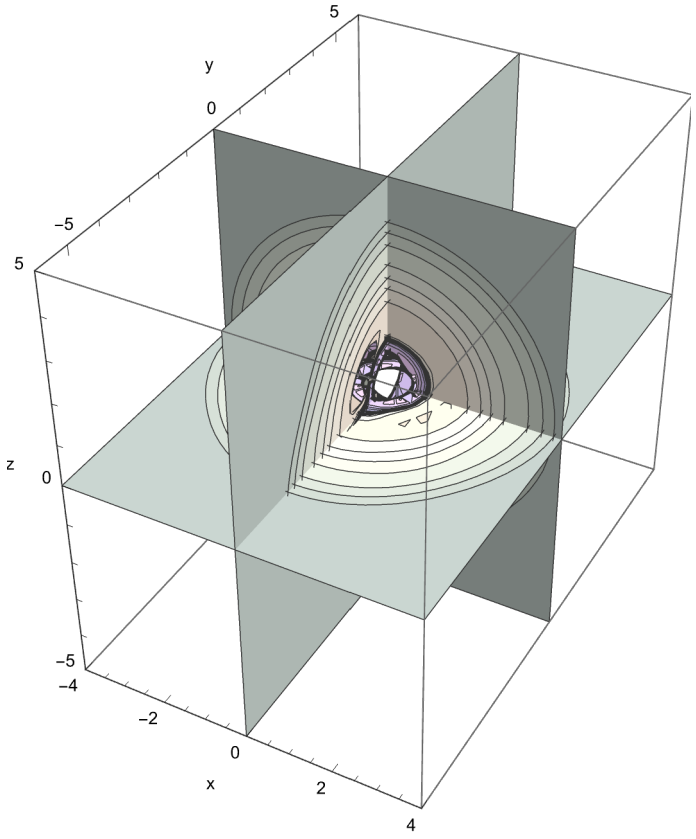
```
    ]
   ]
  ];
gxl[wfn_, {x_?NumericQ, y_?NumericQ, z_?NumericQ}] :=
  Total[First[ElectronDensityDerivative[wfn, {{x, y, z}},
     {
      {3, 0, 0},
      {1, 2, 0},
      {1, 0, 2}
     }]]];
gyl[wfn_, {x_?NumericQ, y_?NumericQ, z_?NumericQ}] :=
  Total[First[ElectronDensityDerivative[wfn, {{x, y, z}},
     {
      {2, 1, 0},
      {0, 3, 0},
      {0, 1, 2}
     }]]];
gzl[wfn_, {x_?NumericQ, y_?NumericQ, z_?NumericQ}] :=
  Total[First[ElectronDensityDerivative[wfn, {{x, y, z}},
     {
      {2, 0, 1},
      {0, 2, 1},
      {0, 0, 3}
     }]]];
gl[wfn_, {x_?NumericQ, y_?NumericQ, z_?NumericQ}] := {
   Total[First[ElectronDensityDerivative[wfn, {{x, y, z}},
      {
       {3, 0, 0},
       {1, 2, 0},
       {1, 0, 2}
      }]]],
   Total[First[ElectronDensityDerivative[w, {{x, y, z}},
      {
       {2, 1, 0},
       {0, 3, 0},
       {0, 1, 2}
      }]]],
   Total[First[ElectronDensityDerivative[w, {{x, y, z}},
      {
       {2, 0, 1},
       {0, 2, 1},
       {0, 0, 3}
      }]]]
  };
```

```
Hl[wfn_, {x_?NumericQ, y_?NumericQ, z_?NumericQ}] := Module[
    {h},
    h = {
      {
       Total[First[ElectronDensityDerivative[wfn, {{x, y, z}},
           {
            {4, 0, 0},
            {2, 2, 0},
            {2, 0, 2}
           }]]],
       Total[First[ElectronDensityDerivative[wfn, {{x, y, z}},
           {
            {3, 1, 0},
            {1, 3, 0},
            {1, 1, 2}
           }]]],
       Total[First[ElectronDensityDerivative[wfn, {{x, y, z}},
           {
            {3, 0, 1},
            {1, 2, 1},
            {1, 0, 3}
           }]]]
      },
      {
       Total[First[ElectronDensityDerivative[wfn, {{x, y, z}},
           {
            {3, 1, 0},
            {1, 3, 0},
            {1, 1, 2}
           }]]],
       Total[First[ElectronDensityDerivative[wfn, {{x, y, z}},
           {
            {2, 2, 0},
            {0, 4, 0},
            {0, 2, 2}
           }]]],
       Total[First[ElectronDensityDerivative[wfn, {{x, y, z}},
           {
            {2, 1, 1},
            {0, 3, 1},
            {0, 1, 3}
           }]]]
      },
      {
```

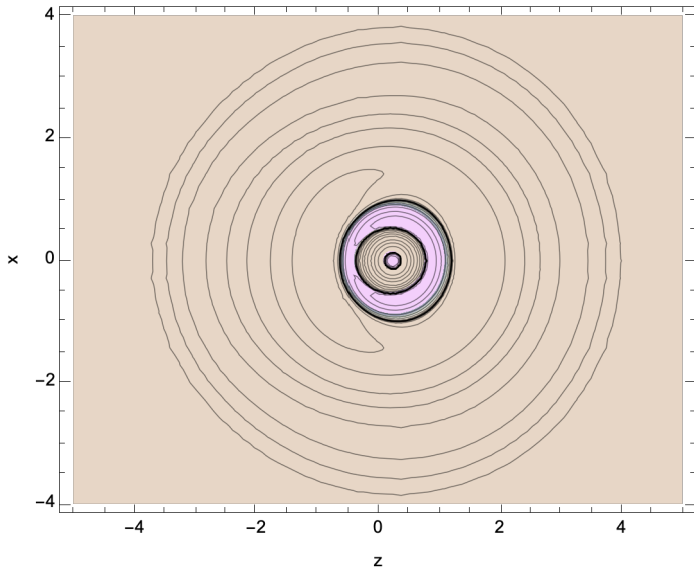
```
      Total[First[ElectronDensityDerivative[wfn, {{x, y, z}},
         {
          {3, 0, 1},
          {1, 2, 1},
          {1, 0, 3}
         }]]],
      Total[First[ElectronDensityDerivative[wfn, {{x, y, z}},
         {
          {2, 1, 1},
          {0, 3, 1},
          {0, 1, 3}
         }]]],
      Total[First[ElectronDensityDerivative[wfn, {{x, y, z}},
         {
          {2, 0, 2},
          {0, 2, 2},
          {0, 0, 4}
         }]]]
     }
    }
  ];
```

A plot of the Laplacian (unfortunately some “tearing” that needs to be resolved):

```
SliceContourPlot3D[
 -laprho[w, {x, y, z}],
 "CenterPlanes",
 {x, bbxmin, bbxmax},
 {y, bbymin, bbymax},
 {z, bbzmin, bbzmax},
 AxesLabel → {"x", "y", "z"},
 Contours → $QTAIMContours,
 PlotRange → {Full, Full, Full, All}
 , BoxRatios → Automatic
 , ColorFunction → "PearlColors"
 , ColorFunctionScaling → True]
```



```
ContourPlot[
 -laprho[w, {x, 0, z}],
 {z, bbzmin, bbzmax},
 {x, bbxmin, bbxmax},
 FrameLabel → {"z", "x"},
 Contours → $QTAIMContours,
 PlotRange → All
 , ColorFunction → "PearlColors"
 , ColorFunctionScaling → False
 , AspectRatio → Automatic]
```



Locate the critical points in the 3D Negative of Laplacian:

```
AbsoluteTiming[
 cpsl = LocateCriticalPoints[
   {
    -gxl[w, {x, y, z}],
    -gyl[w, {x, y, z}],
    -gzl[w, {x, y, z}]
   },
   {x, bbxmin, bbxmax},
   {y, bbymin, bbymax},
   {z, bbzmin, bbzmax}
  ]
]
```

CompiledFunction: Numerical error encountered; proceeding with uncompiled evaluation.

CompiledFunction: Numerical error encountered; proceeding with uncompiled evaluation.

CompiledFunction: Numerical error encountered; proceeding with uncompiled evaluation.

General: Further output of CompiledFunction::cfne will be suppressed during this calculation.

FindRoot: Failed to converge to the requested accuracy or precision within 100 iterations.

FindRoot: Failed to converge to the requested accuracy or precision within 100 iterations.

FindRoot: Failed to converge to the requested accuracy or precision within 100 iterations.

General: Further output of FindRoot::cvmit will be suppressed during this calculation.

$$\{1318.26, \{\{-1.26794, 1.90386\times10^{-16}, 0.941257\},$$
$$\{-1.25166, 0.661657, -0.0103386\}, \{-1.25166, -0.661657, -0.0103386\},$$
$$\{-1.25166, -0.661657, -0.0103386\}, \{-1.25166, -0.661657, -0.0103386\},$$
$$\{-0.846583, 7.92939\times10^{-17}, -0.850825\}, \{-0.66749, 8.92212\times10^{-17}, -0.800568\},$$
$$\{-0.578142, -1.27557\times10^{-18}, 0.438192\}, \{-0.432089, 9.9789\times10^{-18}, -0.786566\},$$
$$\{-0.432089, 4.96707\times10^{-17}, -0.786566\}, \{-0.277579, 1.43229, -1.74983\},$$
$$\{-0.277579, -1.43229, -1.74983\}, \{-0.116007, 1.20832\times10^{-17}, 0.0985677\},$$
$$\{-0.000125282, -0.0358367, 0.380373\}, \{-3.95952\times10^{-6}, 0.0260027, 0.382222\},$$
$$\{-2.56282\times10^{-15}, -0.534134, -0.206893\}, \{-1.97889\times10^{-15}, 2.29192, -0.757691\},$$
$$\{-5.13691\times10^{-16}, 4.02425\times10^{-17}, -0.852028\}, \{-4.81485\times10^{-16}, 1.40626, -1.79305\},$$
$$\{-2.7673\times10^{-16}, -3.71271\times10^{-17}, -1.15279\}, \{-1.8862\times10^{-16}, -1.39918, -0.863799\},$$
$$\{-1.85156\times10^{-16}, 1.193\times10^{-16}, -1.05904\}, \{-9.43756\times10^{-17}, -0.776867, -0.391844\},$$
$$\{-1.79333\times10^{-17}, 0.662261, 0.314336\}, \{-9.3551\times10^{-18}, -1.64277\times10^{-17}, 1.59154\},$$
$$\{-9.3551\times10^{-18}, -1.34588\times10^{-16}, 1.59154\},$$
$$\{-9.3551\times10^{-18}, -8.02169\times10^{-18}, 1.59154\}, \{-9.3551\times10^{-18},$$
$$-8.15996\times10^{-17}, 1.59154\}, \{-5.02467\times10^{-18}, 1.47004\times10^{-18}, 0.846361\},$$
$$\{-6.18696\times10^{-19}, 0.000145544, 0.384262\}, \{-4.07436\times10^{-19}, 2.20429, -1.23547\},$$
$$\{-1.52212\times10^{-19}, 0.955562, 0.696681\}, \{-1.52212\times10^{-19}, 0.955562, 0.696681\},$$
$$\{1.65741\times10^{-20}, 1.39918, -0.863799\}, \{1.65741\times10^{-20}, 1.39918, -0.863799\},$$
$$\{9.58653\times10^{-18}, -0.955562, 0.696681\}, \{1.23755\times10^{-17}, -0.662261, 0.314336\},$$
$$\{1.46191\times10^{-17}, -2.55386\times10^{-17}, -0.45298\},$$
$$\{1.47718\times10^{-16}, -2.29192, -0.757691\}, \{1.61194\times10^{-16}, -2.29192, -0.757691\},$$
$$\{3.99732\times10^{-16}, 0.662261, 0.314336\}, \{8.43477\times10^{-16}, 6.53842\times10^{-17}, -0.45298\},$$
$$\{5.51211\times10^{-6}, -0.0275438, 0.381973\}, \{0.021711, -0.147984, 0.291592\},$$
$$\{0.0589623, 0.531663, -0.204581\}, \{0.0589623, 0.531663, -0.204581\},$$
$$\{0.116007, -4.70364\times10^{-19}, 0.0985677\}, \{0.277579, 1.43229, -1.74983\},$$
$$\{0.277579, -1.43229, -1.74983\}, \{0.578142, -4.98123\times10^{-18}, 0.438192\},$$
$$\{0.578142, 8.48563\times10^{-18}, 0.438192\}, \{1.25166, -0.661657, -0.0103386\},$$
$$\{1.25166, 0.661657, -0.0103386\}, \{1.26794, -1.67238\times10^{-16}, 0.941257\}\}\}$$

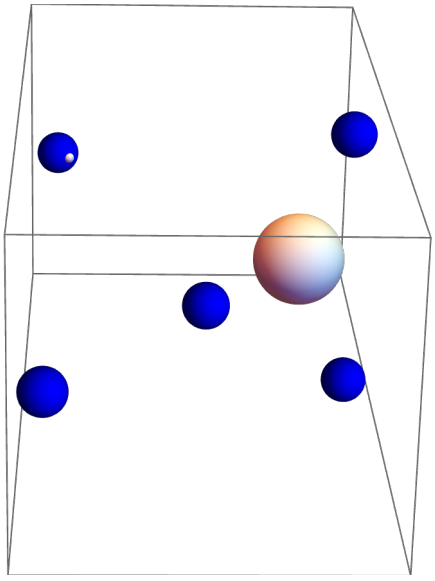
```
CriticalPointRank[{x_, y_, z_}] :=
  Total[Map[If[# == 0, 0, 1] &, Chop[Eigenvalues[-Hl[w, {x, y, z}]]]]];
CriticalPointSignature[{x_, y_, z_}] := Total[
   Map[Which[# < 0, -1, # == 0, 0, # > 0, 1] &, Chop[Eigenvalues[-Hl[w, {x, y, z}]]]]];
```

```
uniqueCpsl = DeleteDuplicates[Chop[cpsl], EuclideanDistance[#1, #2] < 1*^-8 &];

cpslTable = Select[
  Transpose[
   {
    uniqueCpsl[[All, 1]],
    uniqueCpsl[[All, 2]],
    uniqueCpsl[[All, 3]],
    Map[CriticalPointRank, uniqueCpsl],
    Map[CriticalPointSignature, uniqueCpsl]
   }
  ],
  #[[5]] == -3 &
 ]
```

```
{{-0.578142, 0, 0.438192, 3, -3},
 {0, -0.534134, -0.206893, 3, -3}, {0, -1.39918, -0.863799, 3, -3},
 {0, 1.39918, -0.863799, 3, -3}, {0.578142, 0, 0.438192, 3, -3}}
```

Lone Pairs of Electrons on the Oxygen:

```
Graphics3D[
 Join[
  Table[
   {Blue, Sphere[cpslTable[[i, 1 ;; 3]], 0.1]}
   , {i, 1, Length[cpslTable]}]
  ,
  Table[
   Sphere[ncps[[i]], 0.025 * ((w["AtomicNumbers"])[[i]])]
   , {i, 1, Length[ncps]}]
 ]
]
```



Better Bond Path Style Using the (Negative of the) Laplacian of the Electron Density

The sign of the Laplacian at the Bond Critical Point can be used to determine whether a bond path should be solid or dashed. The interpolating functions returned by the steepest ascent algorithm are smooth and show banana-like character in good aesthetic detail. Extract the interpolating functions for the forward/backward bond paths:

```
bondPathInterpolatingFunctions = Table[
   {
    bondPaths[[
      bp, (*bp*)
      1, (*dir*)
      1]],
    Reverse[bondPaths[[
       bp, (*bp*)
       2, (*dir*)
       1]]]
   }
   , {bp, 1, Length[bondPaths]}];
```

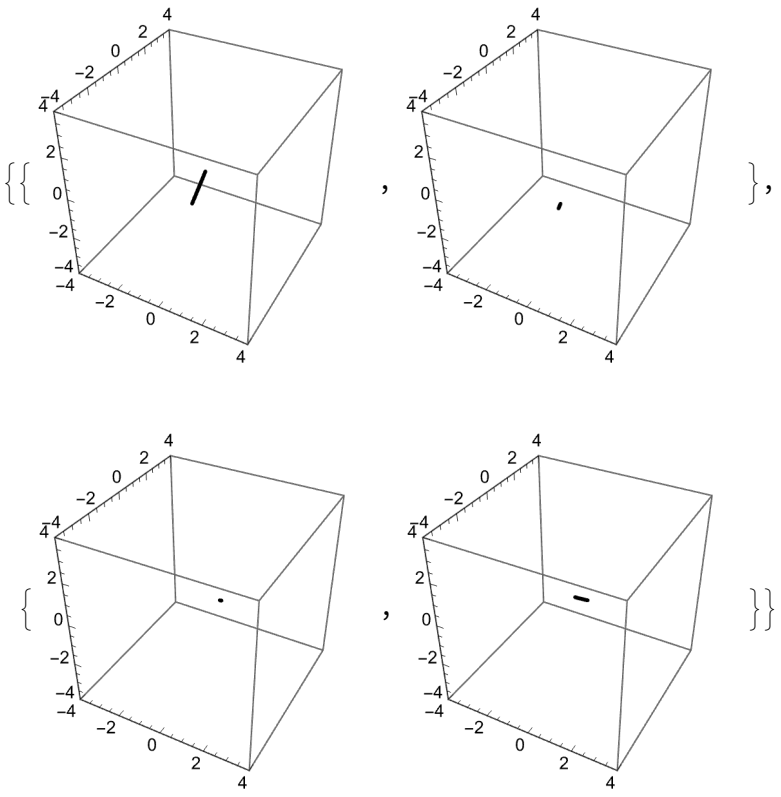
Note the sign of the negative of the Laplacian, we use these to select dashed or solid bond paths:

```
bcpNegativeLaplacians = Map[(-laprho[w, #] &), bcps[[All, 1 ;; 3]]]
```

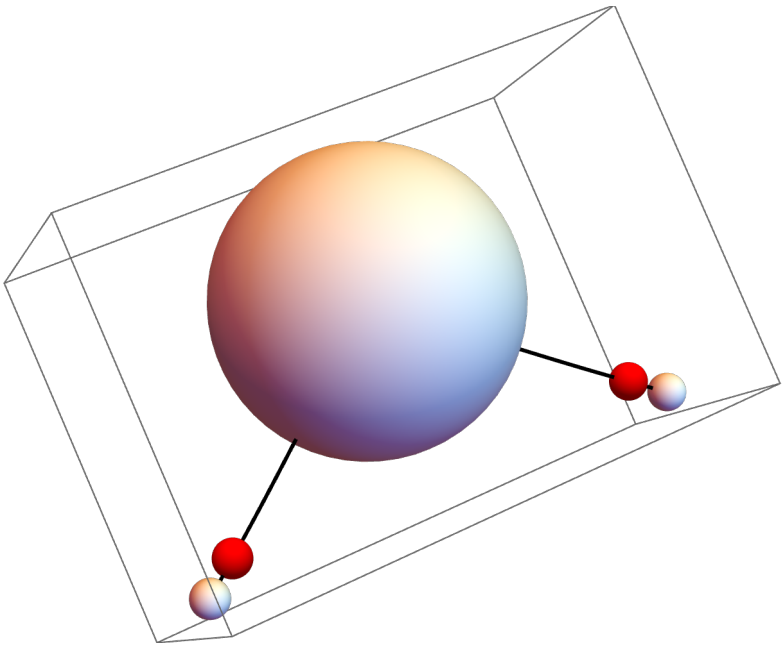
```
{2.55995, 2.55995}
```

Generate the bond paths as ParametricPlot3D functions:

```
bondPathPlots = Table[
  (* backward *)
  s0 = (bondPathInterpolatingFunctions[[bcp, All, All]][[1]])[[0]][[1]][[1]][[1]];
  sf = (bondPathInterpolatingFunctions[[bcp, All, All]][[1]])[[0]][[1]][[1]][[2]];
  forwardPlot = ParametricPlot3D[
    (bondPathInterpolatingFunctions[[bcp, All, All]][[1]]) /. {QTAIM`Private`s → s}
    , {s, s0, sf},
    PlotStyle → If[bcpNegativeLaplacians[[bcp]] > 0, Directive[Black, Thick],
      Directive[Black, Dashed]], PlotRange → {{-4, 4}, {-4, 4}, {-4, 4}}];
  (* forward *)
  s0 = (bondPathInterpolatingFunctions[[bcp, All, All]][[2]])[[0]][[1]][[1]][[1]];
  sf = (bondPathInterpolatingFunctions[[bcp, All, All]][[2]])[[0]][[1]][[1]][[2]];
  backwardPlot = ParametricPlot3D[
    (bondPathInterpolatingFunctions[[bcp, All, All]][[2]]) /. {QTAIM`Private`s → s}
    , {s, s0, sf},
    PlotStyle → If[bcpNegativeLaplacians[[bcp]] > 0, Directive[Black, Thick],
      Directive[Black, Dashed]], PlotRange → {{-4, 4}, {-4, 4}, {-4, 4}}];
  {forwardPlot, backwardPlot}
  , {bcp, 1, Length[bcps]}]
```

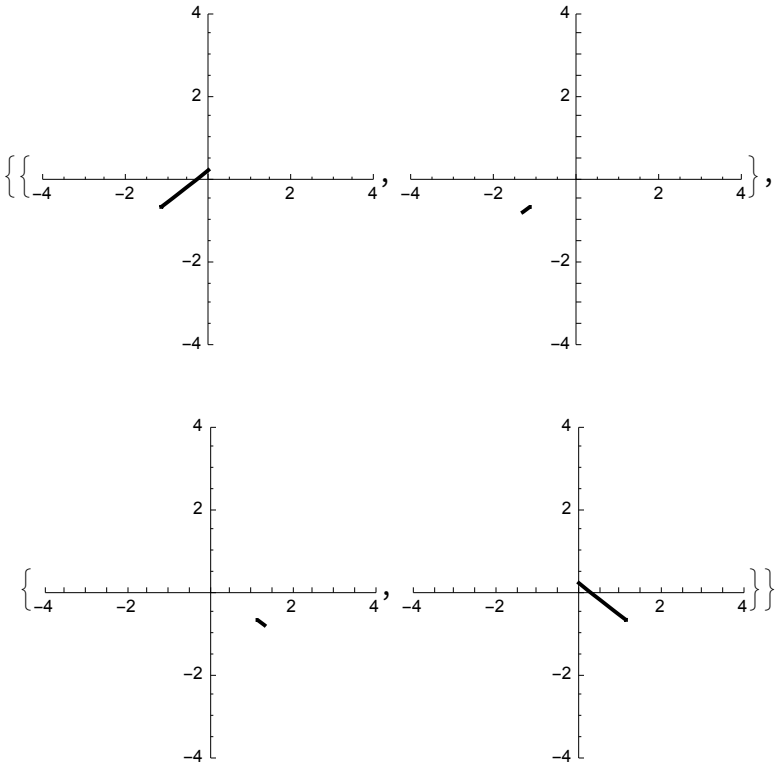


```
graph = Show[
  Graphics3D[
   Join[
    Table[
     {Red, Sphere[bcps[[i, 1 ;; 3]], 0.1]}
     , {i, 1, Length[bcps]}]
    ,
    Table[
     {Green, Sphere[rcps[[i, 1 ;; 3]], 0.1]}
     , {i, 1, Length[rcps]}]
    ,
    Table[
     {Blue, Sphere[ccps[[i, 1 ;; 3]], 0.1]}
     , {i, 1, Length[ccps]}]
    ,
    Table[
     Sphere[ncps[[i]], 0.1 * ((w["AtomicNumbers"])[[i]])]
     , {i, 1, Length[ncps]}]
   ]
  ],
  Flatten[bondPathPlots]
 ]
```

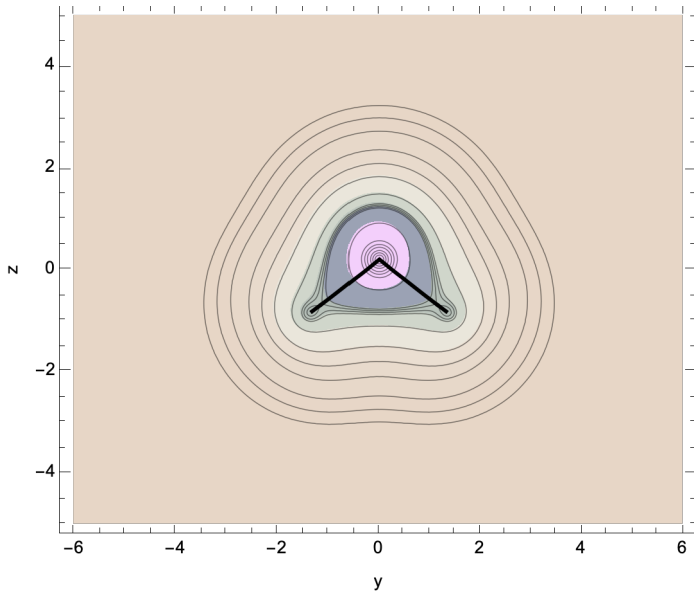


In the plane of the molecule:

```
bondPathPlotLines = Table[
  (* backward *)
  s0 = (bondPathInterpolatingFunctions[[bcp, All, All]][[1]])[[0]][[1]][[1]][[1]];
  sf = (bondPathInterpolatingFunctions[[bcp, All, All]][[1]])[[0]][[1]][[1]][[2]];
  forwardPlot = ParametricPlot[
    (
      (bondPathInterpolatingFunctions[[bcp, All, All]][[1]]) /. {QTAIM`Private`s → s}
     )[[{2, 3}]], {s, s0, sf},
    PlotStyle → If[bcpNegativeLaplacians[[bcp]] > 0, Directive[Black, Thick],
      Directive[Black, Dashed]], PlotRange → {{-4, 4}, {-4, 4}}];
  (* forward *)
  s0 = (bondPathInterpolatingFunctions[[bcp, All, All]][[2]])[[0]][[1]][[1]][[1]];
  sf = (bondPathInterpolatingFunctions[[bcp, All, All]][[2]])[[0]][[1]][[1]][[2]];
  backwardPlot = ParametricPlot[
    (
      (bondPathInterpolatingFunctions[[bcp, All, All]][[2]]) /. {QTAIM`Private`s → s}
     )[[{2, 3}]]
    , {s, s0, sf},
    PlotStyle → If[bcpNegativeLaplacians[[bcp]] > 0, Directive[Black, Thick],
      Directive[Black, Dashed]], PlotRange → {{-4, 4}, {-4, 4}}];
  {forwardPlot, backwardPlot}
  , {bcp, 1, Length[bcps]}]
```



```
Show[
 rhoPlot,
 bondPathPlotLines
]
```



Optimizing the Beta Spheres

```
q = Reap[
    AbsoluteTiming[
     NIntegrate[
      If[rho[w, {x, y, z}] > Min[$QTAIMContoursPositive],
       rho[w, {x, y, z}],
       0
      ],
      {x, bbxmin, bbxmax},
      {y, bbymin, bbymax},
      {z, bbzmin, bbzmax},
      Method → {"LocalAdaptive", "SymbolicProcessing" → 0}
      , AccuracyGoal → 1, PrecisionGoal → Infinity
      , EvaluationMonitor :→ Sow[{x, y, z}]]
    ]
  ];
integrationPoints = q[[2, 1]];
```

Determine the associated nuclear attractor for each point.

```
anas = ParallelMap[
    AssociatedNuclearAttractor[w, ncps, #] &,
    integrationPoints
  ];
```

```
beta0 = Table[
  otherPoints = Select[
     Transpose[
      {
       integrationPoints,
       anas
      }
     ]
     , (#[[2]] ≠ ncp) &][[All, 1]];

  (80 / 100) EuclideanDistance[ncps[[ncp]], First@Nearest[otherPoints, ncps[[ncp]], 1]]

  , {ncp, 1, Length[ncps]}]
```

```
{1.3502, 0.482784, 0.482784}
```

Comparing ODE Integration Methods and the Effects of a better Beta Sphere

Default (Conservative) Beta Sphere Radius

```
AbsoluteTiming[
 anasAutomatic =
   Map[
    AssociatedNuclearAttractor[w, ncps, #, Method → Automatic] &,
    integrationPoints
   ];
]
```

```
{23.6788, Null}
```

```
AbsoluteTiming[
 anasAdams =
   Map[
    AssociatedNuclearAttractor[w, ncps, #, Method → "Adams"] &,
    integrationPoints
   ];
]
```

```
{23.6459, Null}
```

```
AbsoluteTiming[
 anasABM = Map[
    AssociatedNuclearAttractor[w, ncps, #, Method → "ABM"] &,
    integrationPoints
   ];
]
```

```
{87.5545, Null}
```

```
AbsoluteTiming[
 anasBDF = Map[
    AssociatedNuclearAttractor[w, ncps, #, Method → "BDF"] &,
    integrationPoints
   ];
]
```

```
{33.9737, Null}
```

```
AbsoluteTiming[
 anasBS54 = Map[
    AssociatedNuclearAttractor[w, ncps, #, Method → "BS54"] &,
    integrationPoints
   ];
]
```

```
{48.8479, Null}
```

```
AbsoluteTiming[
 anasCashKarp54 = Map[
    AssociatedNuclearAttractor[w, ncps, #, Method → "CashKarp45"] &,
    integrationPoints
   ];
]
```

```
{36.8186, Null}
```

```
AbsoluteTiming[
 anasDOPRI54 = Map[
    AssociatedNuclearAttractor[w, ncps, #, Method → "DOPRI54"] &,
    integrationPoints
   ];
]
```

```
{34.2693, Null}
```

```
AbsoluteTiming[
 anasFehlberg45 = Map[
    AssociatedNuclearAttractor[w, ncps, #, Method → "Fehlberg45"] &,
    integrationPoints
   ];
]
```

```
{30.1987, Null}
```

With Beta Sphere

```
AbsoluteTiming[
 anasAutomaticb0 = Map[
    AssociatedNuclearAttractor[w,
      ncps, #, BetaSphereRadii → beta0, Method → Automatic] &,
    integrationPoints
   ];
]
```

```
{9.07504, Null}
```

```
AbsoluteTiming[
 anasAdamsb0 = Map[
    AssociatedNuclearAttractor[
      w, ncps, #, BetaSphereRadii → beta0, Method → "Adams"] &,
    integrationPoints
   ];
]
```

```
{9.14655, Null}
```

```
AbsoluteTiming[
 anasABMb0 = Map[
    AssociatedNuclearAttractor[
      w, ncps, #, BetaSphereRadii → beta0, Method → "ABM"] &,
    integrationPoints
   ];
]
```

```
{27.8648, Null}
```

```
AbsoluteTiming[
 anasBDFb0 = Map[
    AssociatedNuclearAttractor[
      w, ncps, #, BetaSphereRadii → beta0, Method → "BDF"] &,
    integrationPoints
   ];
]
```

```
{13.1608, Null}
```

```
AbsoluteTiming[
 anasBS54b0 = Map[
    AssociatedNuclearAttractor[
      w, ncps, #, BetaSphereRadii → beta0, Method → "BS54"] &,
    integrationPoints
   ];
]
```

```
{36.0573, Null}
```

```
AbsoluteTiming[
 anasCashKarp54b0 = Map[
    AssociatedNuclearAttractor[w, ncps,
      #, BetaSphereRadii → beta0, Method → "CashKarp45"] &,
    integrationPoints
   ];
]
```

```
{25.1615, Null}
```

```
AbsoluteTiming[
 anasDOPRI54b0 = Map[
    AssociatedNuclearAttractor[w,
      ncps, #, BetaSphereRadii → beta0, Method → "DOPRI54"] &,
    integrationPoints
   ];
]
```

```
{23.1498, Null}
```

```
AbsoluteTiming[
 anasFehlberg45b0 = Map[
    AssociatedNuclearAttractor[w, ncps,
      #, BetaSphereRadii → beta0, Method → "Fehlberg45"] &,
    integrationPoints
   ];
]
```

```
{18.5581, Null}
```

Plots of Atomic Basins

This package defines a function called AssociatedNuclearAttractor that returns the index of the nuclear critical point that is the terminus of the steepest ascent path from a test point. This is arguably the most “expensive” part of QTAIM: every region of space needs to be tested to ascertain to which NCP it belongs. That could be hundreds of gradient evaluations for even one nominal function evaluation!

AssociatedNuclearAttractor

```
AssociatedNuclearAttractor[w, ncps, {0., 0., 0.}]
```

```
1
```

```
AssociatedNuclearAttractor[w, ncps, {0., 0., 0.}, BetaSphereRadii → beta0]
```

```
1
```

```
AssociatedNuclearAttractor[w, ncps, {0., 2., -2.}, BetaSphereRadii → beta0]
```

```
2
```

```
AssociatedNuclearAttractor[w, ncps, {0., -2., -2.}, BetaSphereRadii → beta0]
```

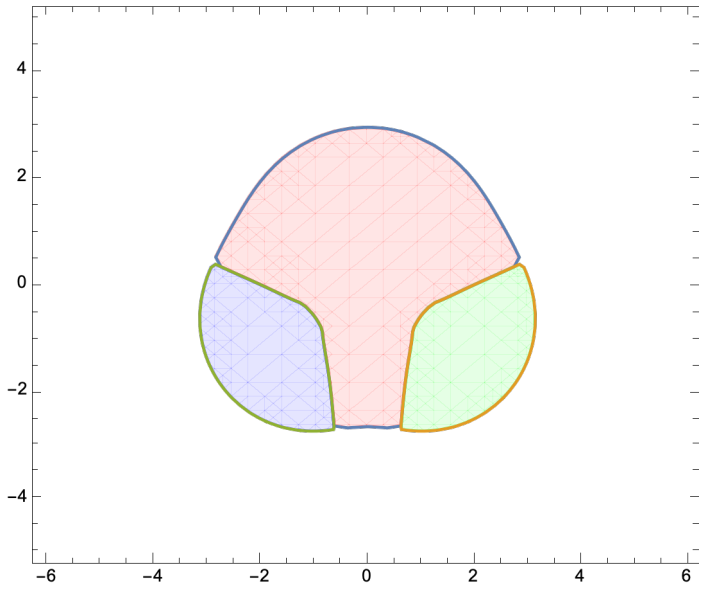
```
3
```

```
AssociatedNuclearAttractor[w, ncps, {2., 2., 2.}]
```

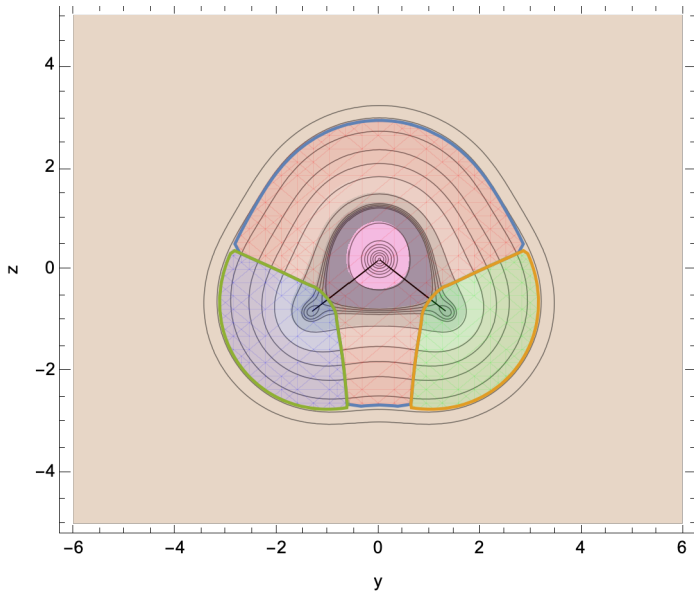
```
0
```

Plots

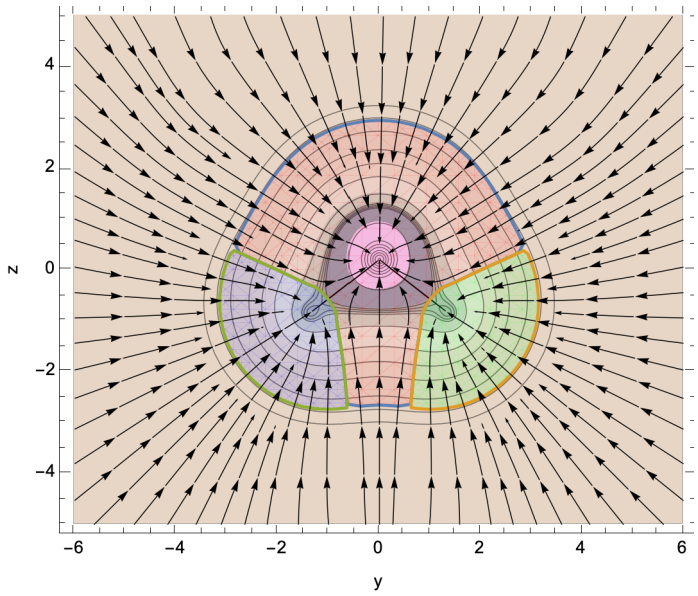
```
rp = RegionPlot[
  {
   AssociatedNuclearAttractor[w, ncps, {0., y, z}, BetaSphereRadii → beta0] == 1,
   AssociatedNuclearAttractor[w, ncps, {0., y, z}, BetaSphereRadii → beta0] == 2,
   AssociatedNuclearAttractor[w, ncps, {0., y, z}, BetaSphereRadii → beta0] == 3
  },
  {y, bbymin, bbymax},
  {z, bbzmin, bbzmax}
  , PlotStyle → {
    Directive[Red, Opacity[0.10]],
    Directive[Green, Opacity[0.10]],
    Directive[Blue, Opacity[0.10]]
   }
  , AspectRatio → Automatic]
```



```
Show[
 rhoPlot,
 Graphics[bondPathLines],
 rp
]
```

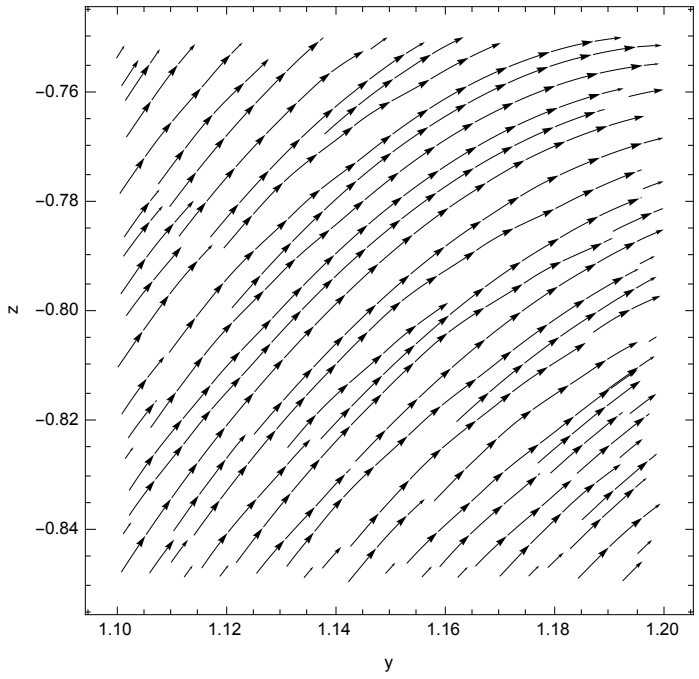



```
Show[
 rhoPlot,
 Graphics[bondPathLines],
 rp,
 streamPlot
]
```



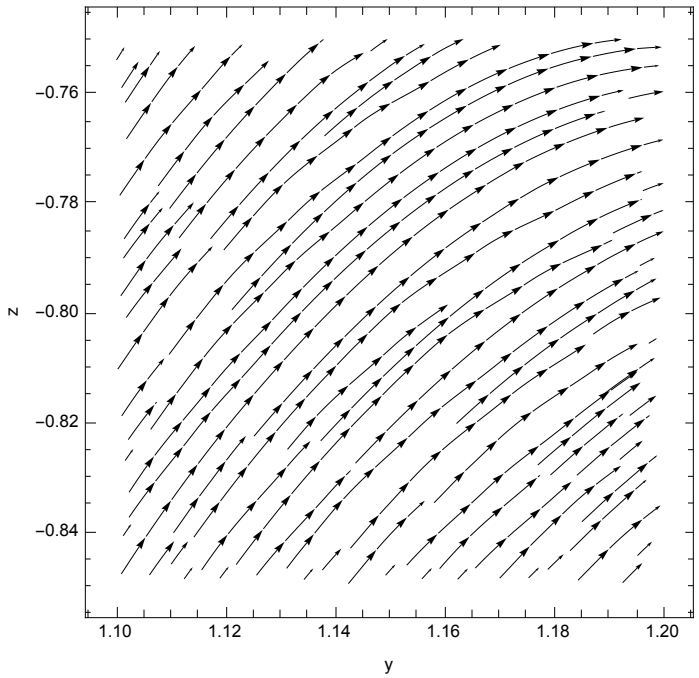
Zoom in on the detail of the one of the bond critical points:

```
streamPlotZoom = StreamPlot[
  gyz[w, {y, z}],
  {y, 1.10, 1.20},
  {z, -0.85, -0.75},
  FrameLabel → {"y", "z"}
  , StreamStyle → Black
  , StreamColorFunction → None
  , AspectRatio → Automatic]
```



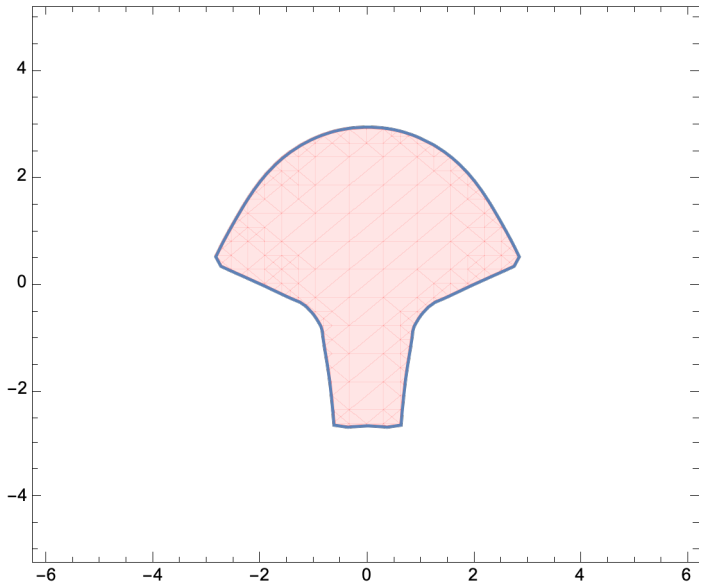
Annotate with bond critical point:

```
Show[
 streamPlotZoom,
 Graphics[{Red, Disk[bcps[[1, 2 ;; 3]], 0.0025]}]
]
```

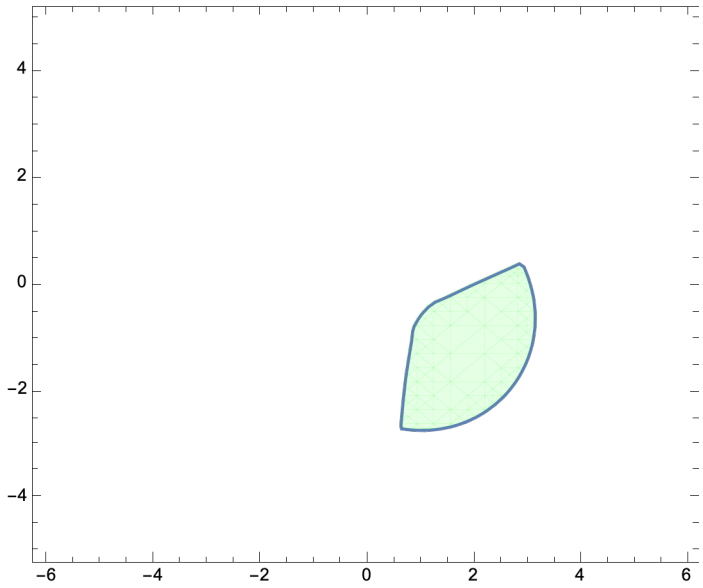


One of the purposes of this package is to facilitate graphics and plots. Here, it is shown how to draw individual components so that only the necessary regions are plotted. (Otherwise, graphics get complicated and occluded.)

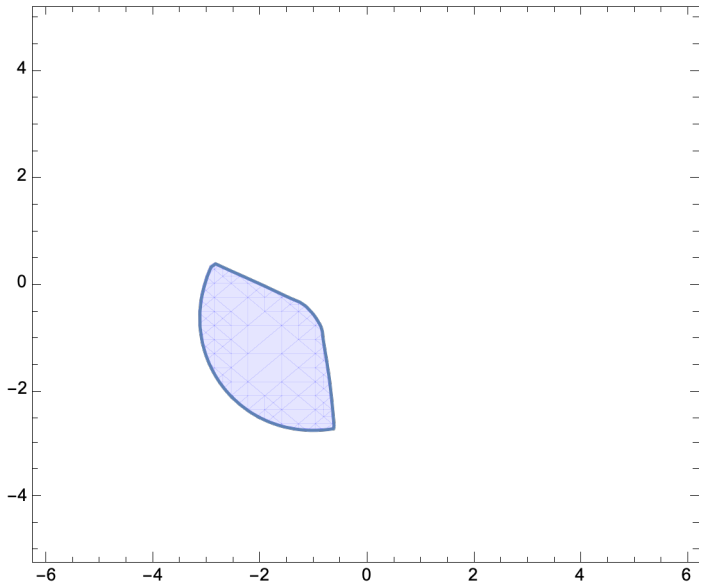
```
rpO = RegionPlot[
  AssociatedNuclearAttractor[w, ncps, {0., y, z}, BetaSphereRadii → beta0] == 1,
  {y, bbymin, bbymax},
  {z, bbzmin, bbzmax}
  , PlotStyle → Directive[Red, Opacity[0.10]]
  , AspectRatio → Automatic
 ]
```



```
rpH1 = RegionPlot[
  AssociatedNuclearAttractor[w, ncps, {0., y, z}, BetaSphereRadii → beta0] == 2,
  {y, bbymin, bbymax},
  {z, bbzmin, bbzmax}
  , PlotStyle → Directive[Green, Opacity[0.10]]
  , AspectRatio → Automatic
 ]
```

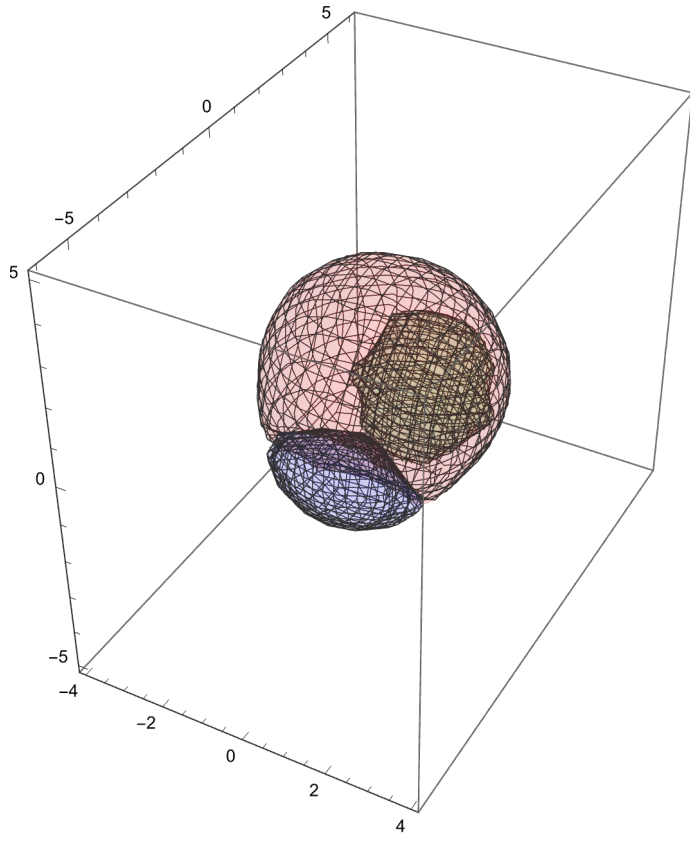


```
rpH2 = RegionPlot[
  AssociatedNuclearAttractor[w, ncps, {0., y, z}, BetaSphereRadii → beta0] == 3,
  {y, bbymin, bbymax},
  {z, bbzmin, bbzmax}
  , PlotStyle → Directive[Blue, Opacity[0.10]]
  , AspectRatio → Automatic
 ]
```

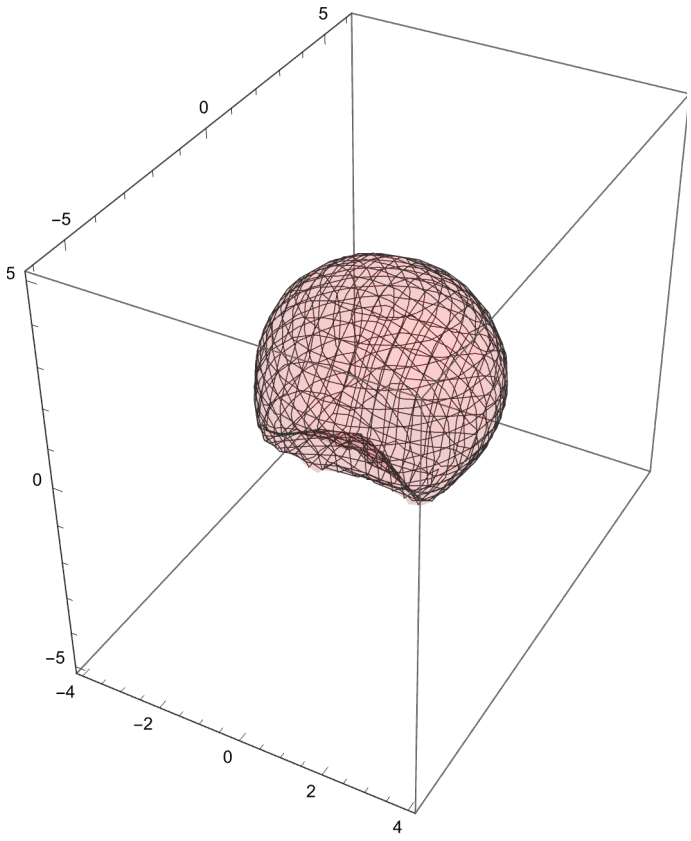


3D Basin plots can be generated in the same way. Note that increasing PlotPoints makes them smoother, with an increase in computer time.

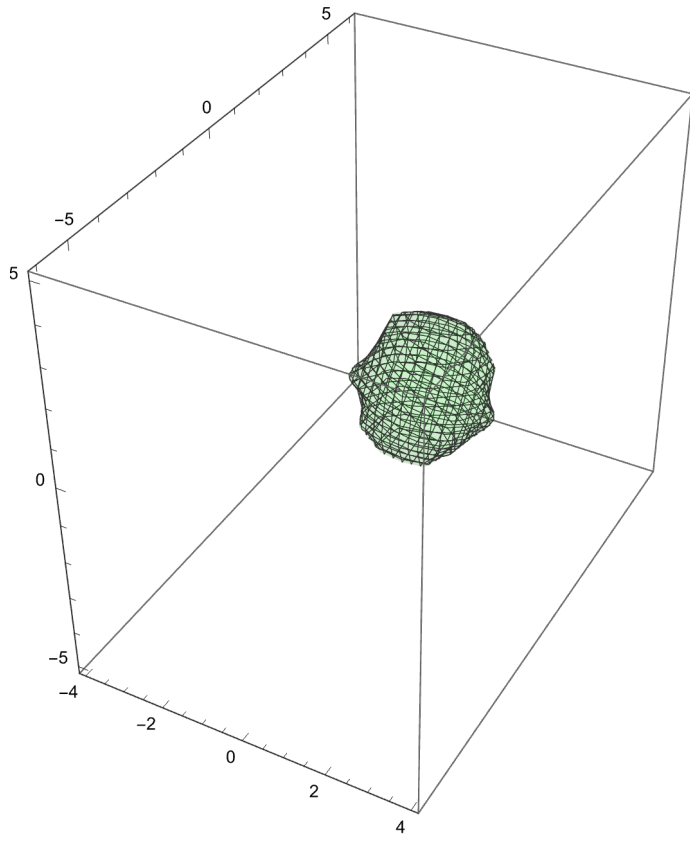
```
rp3D = RegionPlot3D[
  {
   AssociatedNuclearAttractor[w, ncps, {x, y, z}, BetaSphereRadii → beta0] == 1,
   AssociatedNuclearAttractor[w, ncps, {x, y, z}, BetaSphereRadii → beta0] == 2,
   AssociatedNuclearAttractor[w, ncps, {x, y, z}, BetaSphereRadii → beta0] == 3
  },
  {x, bbxmin, bbxmax},
  {y, bbymin, bbymax},
  {z, bbzmin, bbzmax}
  , PlotStyle → {
    Directive[Red, Opacity[0.10]],
    Directive[Green, Opacity[0.10]],
    Directive[Blue, Opacity[0.10]]
   }
  , BoxRatios → Automatic
 ]
```



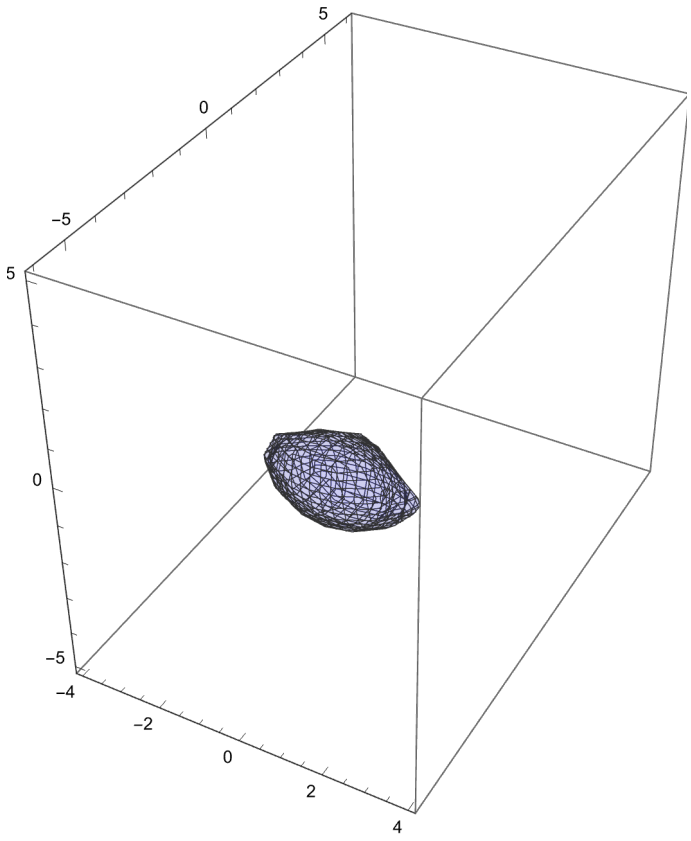
```
rp3DO = RegionPlot3D[
  AssociatedNuclearAttractor[w, ncps, {x, y, z}, BetaSphereRadii → beta0] == 1,
  {x, bbxmin, bbxmax},
  {y, bbymin, bbymax},
  {z, bbzmin, bbzmax}
  , PlotStyle → Directive[Red, Opacity[0.10]]
  , BoxRatios → Automatic
 ]
```



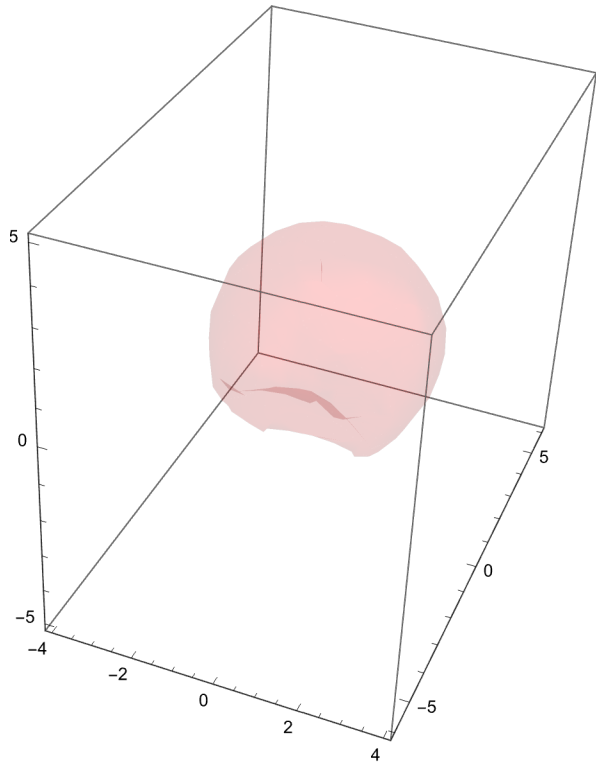
```
rp3DH1 = RegionPlot3D[
  AssociatedNuclearAttractor[w, ncps, {x, y, z}, BetaSphereRadii → beta0] == 2,
  {x, bbxmin, bbxmax},
  {y, bbymin, bbymax},
  {z, bbzmin, bbzmax}
  , PlotStyle → Directive[Green, Opacity[0.10]]
  , BoxRatios → Automatic
 ]
```



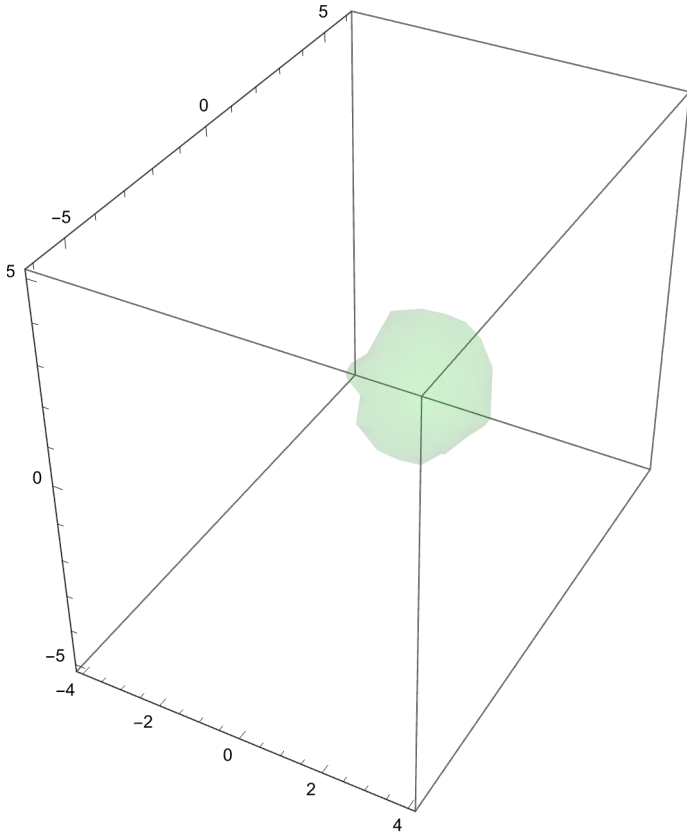
```
rp3DH2 = RegionPlot3D[
  AssociatedNuclearAttractor[w, ncps, {x, y, z}, BetaSphereRadii → beta0] == 3,
  {x, bbxmin, bbxmax},
  {y, bbymin, bbymax},
  {z, bbzmin, bbzmax}
  , PlotStyle → Directive[Blue, Opacity[0.10]]
  , BoxRatios → Automatic
 ]
```



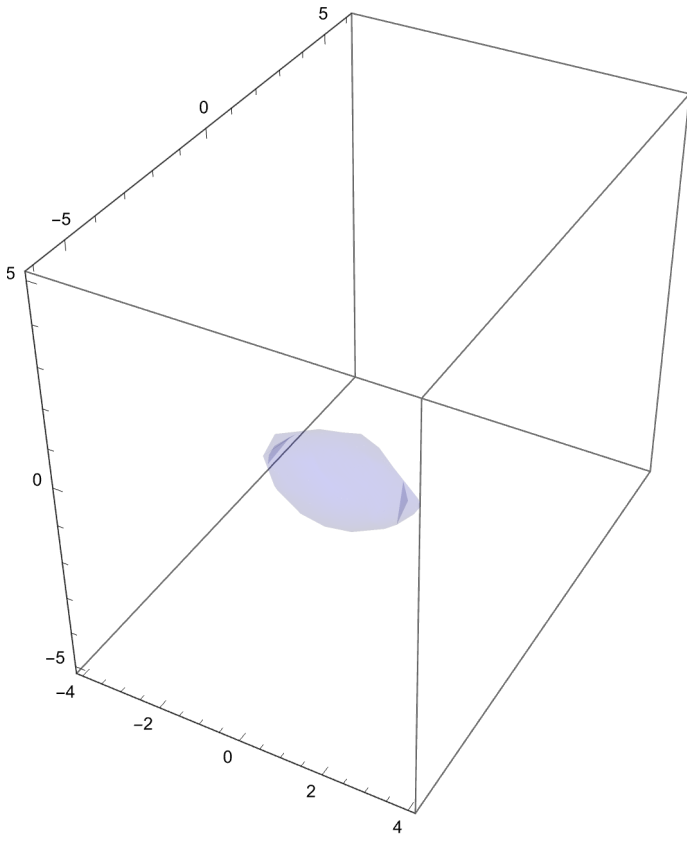
```
rp3DOnomesh = RegionPlot3D[
  AssociatedNuclearAttractor[w, ncps, {x, y, z}, BetaSphereRadii → beta0] == 1,
  {x, bbxmin, bbxmax},
  {y, bbymin, bbymax},
  {z, bbzmin, bbzmax}
  , PlotStyle → Directive[Red, Opacity[0.10]]
  , Mesh → None
  , BoxRatios → Automatic
 ]
```



```
rp3DH1nomesh = RegionPlot3D[
  AssociatedNuclearAttractor[w, ncps, {x, y, z}, BetaSphereRadii → beta0] == 2,
  {x, bbxmin, bbxmax},
  {y, bbymin, bbymax},
  {z, bbzmin, bbzmax}
  , PlotStyle → Directive[Green, Opacity[0.10]]
  , Mesh → None
  , BoxRatios → Automatic
 ]
```



```
rp3DH2nomesh = RegionPlot3D[
  AssociatedNuclearAttractor[w, ncps, {x, y, z}, BetaSphereRadii → beta0] == 3,
  {x, bbxmin, bbxmax},
  {y, bbymin, bbymax},
  {z, bbzmin, bbzmax}
  , PlotStyle → Directive[Blue, Opacity[0.10]]
  , Mesh → None
  , BoxRatios → Automatic
 ]
```



Atomic Properties with Integration over Basins in Cartesian Space

(Note that AccuracyGoals are for demo purposes only. Some analyses require increased accuracy/precision.)

It turns out beneficial to develop an integration strategy prior to integrating the irregular atomic basins. Here, we compare what Automatic gives, GlobalAdaptive, vs. LocalAdaptive. Please note these integrations are in Cartesian space and not in e.g. traditional Spherical Polar coordinates as rays shooting out, starting on a nuclear attractor. The hope is to use adaptive integration where needed in x,y, and z, and avoid strong orientation effects in spherical polar coordinates. (These pervade many

plots in QTAIM.) There is a different type of mis-specification in Lebedev integration: seldom does one want perfect spherical integration either! Cartesian space representation does not suffer from unreachability problems, such as curved surfaces that rays can not reach, though a steepest ascent path exists. Let's adapt in Cartesian space if possible, and reserve comparisons for a later date, especially adaptive spherical polar integration.

Long story short: Choose LocalAdaptive.

The whole system:

```
q = AbsoluteTiming[
  NIntegrate[
   If[rho[w, {x, y, z}] > Min[$QTAIMContoursPositive],
    rho[w, {x, y, z}],
    0
   ],
   {x, bbxmin, bbxmax},
   {y, bbymin, bbymax},
   {z, bbzmin, bbzmax},
   Method → {Automatic, "SymbolicProcessing" → 0}
   , AccuracyGoal → 4, PrecisionGoal → Infinity]
 ]
```

NIntegrate: Numerical integration converging too slowly; suspect one of the following: singularity, value of the integration is 0, highly oscillatory integrand, or WorkingPrecision too small.

NIntegrate: The global error of the strategy GlobalAdaptive has increased more than 2000 times. The global error is expected to decrease monotonically after a number of integrand evaluations. Suspect one of the following: the working precision is insufficient for the specified precision goal; the integrand is highly oscillatory or it is not a (piecewise) smooth function; or the true value of the integral is 0. Increasing the value of the GlobalAdaptive option MaxErrorIncreases might lead to a convergent numerical integration. NIntegrate obtained 9.896484095206976` and 0.003343720127204763` for the integral and error estimates.

```
{6.68422, 9.89648}
```

```
q = AbsoluteTiming[
  NIntegrate[
   If[rho[w, {x, y, z}] > Min[$QTAIMContoursPositive],
    rho[w, {x, y, z}],
    0
   ],
   {x, bbxmin, bbxmax},
   {y, bbymin, bbymax},
   {z, bbzmin, bbzmax},
   Method → {"GlobalAdaptive", "SymbolicProcessing" → 0}
   , AccuracyGoal → 4, PrecisionGoal → Infinity]
 ]
```

NIntegrate: Numerical integration converging too slowly; suspect one of the following: singularity, value of the integration is 0, highly oscillatory integrand, or WorkingPrecision too small.

NIntegrate: The global error of the strategy GlobalAdaptive has increased more than 2000 times. The global error is expected to decrease monotonically after a number of integrand evaluations. Suspect one of the following: the working precision is insufficient for the specified precision goal; the integrand is highly oscillatory or it is not a (piecewise) smooth function; or the true value of the integral is 0. Increasing the value of the GlobalAdaptive option MaxErrorIncreases might lead to a convergent numerical integration. NIntegrate obtained 9.896484095206976` and 0.003343720127204763` for the integral and error estimates.

```
{6.62661, 9.89648}
```

LocalAdaptive is much faster than GlobalAdaptive, Automatic gets it wrong every time. I wonder if this is because it can re-use expensive function evaluations! This is contrary to what I expected, since GlobalAdaptive is usually superior:

```
q = AbsoluteTiming[
  NIntegrate[
   If[rho[w, {x, y, z}] > Min[$QTAIMContoursPositive],
    rho[w, {x, y, z}],
    0
   ],
   {x, bbxmin, bbxmax},
   {y, bbymin, bbymax},
   {z, bbzmin, bbzmax},
   Method → {"LocalAdaptive", "SymbolicProcessing" → 0}
   , AccuracyGoal → 4, PrecisionGoal → Infinity]
 ]
```

```
{2.60652, 9.89807}
```

LocalAdaptive is ten times faster!

Parallel Processing:

```
xrange = Subdivide[bbxmin, bbxmax, 2];
yrange = Subdivide[bbymin, bbymax, 2];
zrange = Subdivide[bbzmin, bbzmax, 2];

q = AbsoluteTiming[
  ParallelSum[
   NIntegrate[
    rho[w, {x, y, z}],
    {x, xrange[[ix]], xrange[[ix + 1]]},
    {y, xrange[[iy]], xrange[[iy + 1]]},
    {z, xrange[[iz]], xrange[[iz + 1]]},
    Method → {"LocalAdaptive", "SymbolicProcessing" → 0}
    , AccuracyGoal → 4, PrecisionGoal → Infinity
   ]
   , {ix, Length[xrange] - 1}
   , {iy, Length[yrange] - 1}
   , {iz, Length[zrange] - 1}
  ]
 ]
{0.265053, 9.99639}
```

(Slower, but likely due to overhead for a pretty small molecule subdivided into many pieces. Should also increase accuracy since error is additive over these subdivided regions!)

Let's try a different way, employing two mirror-plane symmetry:

```
q = AbsoluteTiming[
  4 * NIntegrate[
    If[rho[w, {x, y, z}] > Min[$QTAIMContoursPositive],
     rho[w, {x, y, z}],
     0
    ],
    {x, 0, bbxmax},
    {y, 0, bbymax},
    {z, bbzmin, bbzmax},
    Method → {Automatic, "SymbolicProcessing" → 0}
    , AccuracyGoal → 4, PrecisionGoal → Infinity]
 ]
```

NIntegrate: Numerical integration converging too slowly; suspect one of the following: singularity, value of the integration is 0, highly oscillatory integrand, or WorkingPrecision too small.

NIntegrate: The global error of the strategy GlobalAdaptive has increased more than 2000 times. The global error is expected to decrease monotonically after a number of integrand evaluations. Suspect one of the following: the working precision is insufficient for the specified precision goal; the integrand is highly oscillatory or it is not a (piecewise) smooth function; or the true value of the integral is 0. Increasing the value of the GlobalAdaptive option MaxErrorIncreases might lead to a convergent numerical integration. NIntegrate obtained 2.4740506064393326` and 0.0012874334067809474` for the integral and error estimates.

```
{4.54053, 9.8962}
```

```
q = AbsoluteTiming[
  4 * NIntegrate[
    If[rho[w, {x, y, z}] > Min[$QTAIMContoursPositive],
     rho[w, {x, y, z}],
     0
    ],
    {x, 0, bbxmax},
    {y, 0, bbymax},
    {z, bbzmin, bbzmax},
    Method → {"GlobalAdaptive", "SymbolicProcessing" → 0}
    , AccuracyGoal → 4, PrecisionGoal → Infinity]
 ]
```

NIntegrate: Numerical integration converging too slowly; suspect one of the following: singularity, value of the integration is 0, highly oscillatory integrand, or WorkingPrecision too small.

NIntegrate: The global error of the strategy GlobalAdaptive has increased more than 2000 times. The global error is expected to decrease monotonically after a number of integrand evaluations. Suspect one of the following: the working precision is insufficient for the specified precision goal; the integrand is highly oscillatory or it is not a (piecewise) smooth function; or the true value of the integral is 0. Increasing the value of the GlobalAdaptive option MaxErrorIncreases might lead to a convergent numerical integration. NIntegrate obtained 2.4740506064393326` and 0.0012874334067809474` for the integral and error estimates.

```
{4.55082, 9.8962}
```

```
q = AbsoluteTiming[
  4 * NIntegrate[
    If[rho[w, {x, y, z}] > Min[$QTAIMContoursPositive],
     rho[w, {x, y, z}],
     0
    ],
    {x, 0, bbxmax},
    {y, 0, bbymax},
    {z, bbzmin, bbzmax},
    Method → {"LocalAdaptive", "SymbolicProcessing" → 0}
    , AccuracyGoal → 4, PrecisionGoal → Infinity]
 ]
```

```
{0.576972, 9.89556}
```

Shocking time difference, same integrated charge.

Alternative definition of the envelope, when Associated Nuclear Attractor is not 0. This is probably more like what an atomic basin integration would be like, at least for exterior atoms--terminating by default at the early condition where if ρ<(x0, y0, z0) < Min[$QTAIMContourPositive] then assigned ncp index 0:

Let's try with LocalAdaptive (skipping GlobalAdaptive this time) for the whole molecule:

```
qalt = AbsoluteTiming[
  NIntegrate[
   If[AssociatedNuclearAttractor[w, ncps, {x, y, z}, BetaSphereRadii → beta0] ≠ 0,
    rho[w, {x, y, z}],
    0
   ],
   {x, bbxmin, bbxmax},
   {y, bbymin, bbymax},
   {z, bbzmin, bbzmax},
   Method → {"LocalAdaptive", "SymbolicProcessing" → 0}
   , AccuracyGoal → 4, PrecisionGoal → Infinity]
 ]
```

```
{537.891, 9.78388}
```

It's a good thing that we spent some time investigating integration technique before tackling harder problems.

Integrate over individual atomic basins, using LocalAdaptive. It could mean orders of magnitude time difference for an expensive integration:

(Note we can help the integration to locate regions of non-zero density by placing nuclear critical point coordinates in the integration limits.)

```
qO = AbsoluteTiming[
  With[{ncp = 1},
   4 * NIntegrate[
     If[
      AssociatedNuclearAttractor[w, ncps, {x, y, z}, BetaSphereRadii → beta0] == ncp,
      rho[w, {x, y, z}],
      0],
     {x, 0, bbxmax},
     {y, 0, bbymax},
     {z, bbzmin, ncps[[ncp, 3]], bbzmax},
     Method → {"LocalAdaptive", "SymbolicProcessing" → 0}
     , AccuracyGoal → 4, PrecisionGoal → Infinity]
  ]
 ]
```

```
{214.32, 8.83892}
```

```
qH1andH2 = AbsoluteTiming[
  With[{ncp = 2},
   4 * NIntegrate[
     If[
      AssociatedNuclearAttractor[w, ncps, {x, y, z}, BetaSphereRadii → beta0] == ncp,
      rho[w, {x, y, z}],
      0],
     {x, 0, bbxmax},
     {y, 0, ncps[[ncp, 2]], bbymax},
     {z, bbzmin, ncps[[ncp, 3]], bbzmax},
     Method → {"LocalAdaptive", "SymbolicProcessing" → 0}
     , AccuracyGoal → 4, PrecisionGoal → Infinity]
  ]
 ]
```

```
{88.194, 0.583129}
```

Total atomic charge as sum of components:

```
(qO + qH1andH2)
```

```
{302.514, 9.42205}
```

Difference between whole molecule and fragments:

```
Last[qalt] - (Last[qO] + Last[qH1andH2])
```

```
0.361833
```

Integrate the Laplacian (should be zeros for each integration, by zero-flux condition). We will use the LocalAdaptive strategy, but it is really a different type of integral and we should come back and compare with GlobalAdaptive/Automatic. It is known that there are problems integrating to values that are nearly 0, and so some warnings may appear. Traditionally this is used as a check for the quality of the basin delineation, but here all grids are different because all integrands are different and adaptively sampled with regard to basin and integral value!

```
L = AbsoluteTiming[
  NIntegrate[
   If[rho[w, {x, y, z}] > Min[$QTAIMContoursPositive],
    laprho[w, {x, y, z}],
    0
   ],
   {x, bbxmin, bbxmax},
   {y, bbymin, bbymax},
   {z, bbzmin, bbzmax},
   Method → {"LocalAdaptive", "SymbolicProcessing" → 0}
   , AccuracyGoal → 4, PrecisionGoal → Infinity]
 ]
```

```
{2906.59, -0.747796}
```

The integrated Laplacian density should be zero if truly a quantum (sub)system bound by the zero-flux condition:

```
LO = AbsoluteTiming[
  With[{ncp = 1},
   4 * NIntegrate[
     If[
      AssociatedNuclearAttractor[w, ncps, {x, y, z}, BetaSphereRadii → beta0] == ncp,
      laprho[w, {x, y, z}],
      0],
     {x, 0, bbxmax},
     {y, 0, bbymax},
     {z, bbzmin, bbzmax},
     Method → {"LocalAdaptive", "SymbolicProcessing" → 0}
     , AccuracyGoal → 4, PrecisionGoal → Infinity]
  ]
 ]
```

```
{1773.27, -1.06619}
```

```
LH1andH2 = AbsoluteTiming[
  With[{ncp = 2},
   4 * NIntegrate[
     If[
      AssociatedNuclearAttractor[w, ncps, {x, y, z}, BetaSphereRadii → beta0] == ncp,
      laprho[w, {x, y, z}],
      0],
     {x, 0, bbxmax},
     {y, 0, bbymax},
     {z, bbzmin, bbzmax},
     Method → {"LocalAdaptive", "SymbolicProcessing" → 0}
     , AccuracyGoal → 4, PrecisionGoal → Infinity]
  ]
 ]
```

```
{737.908, -0.29405}
```

```
(LO + LH1andH2)
```

```
{2511.17, -1.36024}
```

```
Last[L] - (Last[LO] + Last[LH1andH2])
```

```
0.612442
```

Volume

```
v = AbsoluteTiming[
  NIntegrate[
   If[rho[w, {x, y, z}] > Min[$QTAIMContoursPositive],
    1,
    0
   ],
   {x, bbxmin, bbxmax},
   {y, bbymin, bbymax},
   {z, bbzmin, bbzmax},
   Method → {"LocalAdaptive", "SymbolicProcessing" → 0}
   , AccuracyGoal → 4, PrecisionGoal → Infinity]
 ]
```

```
{84.1175, 136.477}
```

```
vO = AbsoluteTiming[
  With[{ncp = 1},
   4 * NIntegrate[
     If[
      AssociatedNuclearAttractor[w, ncps, {x, y, z}, BetaSphereRadii → beta0] == ncp,
      1,
      0],
     {x, 0, bbxmax},
     {y, 0, bbymax},
     {z, bbzmin, ncps[[ncp, 3]], bbzmax},
     Method → {"LocalAdaptive", "SymbolicProcessing" → 0}
     , AccuracyGoal → 2, PrecisionGoal → Infinity]
  ]
 ]
```

```
{148.983, 75.3168}
```

```
vH1andH2 = AbsoluteTiming[
  With[{ncp = 2},
   4 * NIntegrate[
     If[
      AssociatedNuclearAttractor[w, ncps, {x, y, z}, BetaSphereRadii → beta0] == ncp,
      1,
      0],
     {x, 0, bbxmax},
     {y, 0, ncps[[ncp, 2]], bbymax},
     {z, bbzmin, ncps[[ncp, 3]], bbzmax},
     Method → {"LocalAdaptive", "SymbolicProcessing" → 0}
     , AccuracyGoal → 2, PrecisionGoal → Infinity]
  ]
 ]
```

```
{69.9489, 17.328}
```

```
vO + 2 * vH1andH2
```

```
{288.88, 109.973}
```

Spherical Polar Basin Delineation and Integration

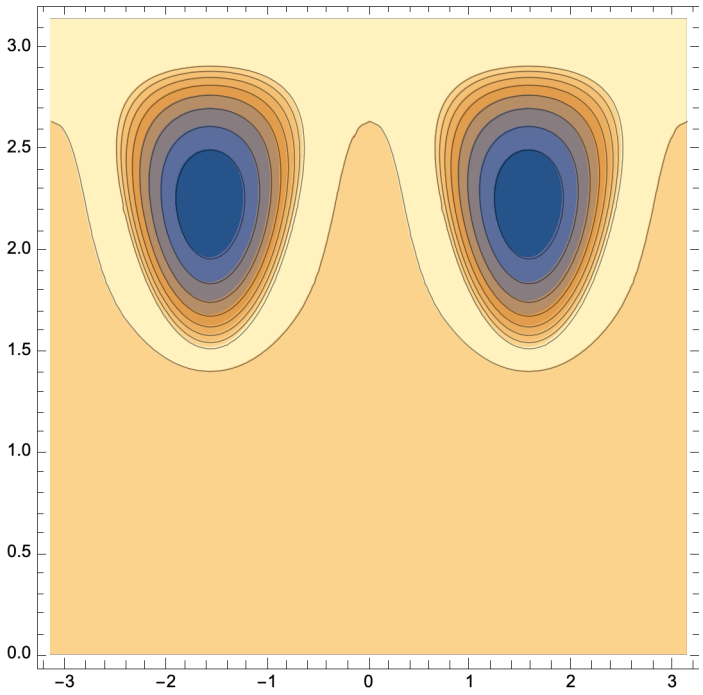
Interatomic Surface Radius Plot

InteratomicSurface is a function that returns the {position, radius, and associated nuclear critical points} at a parametric ray shooting from a nuclear critical point along θ and ϕ.

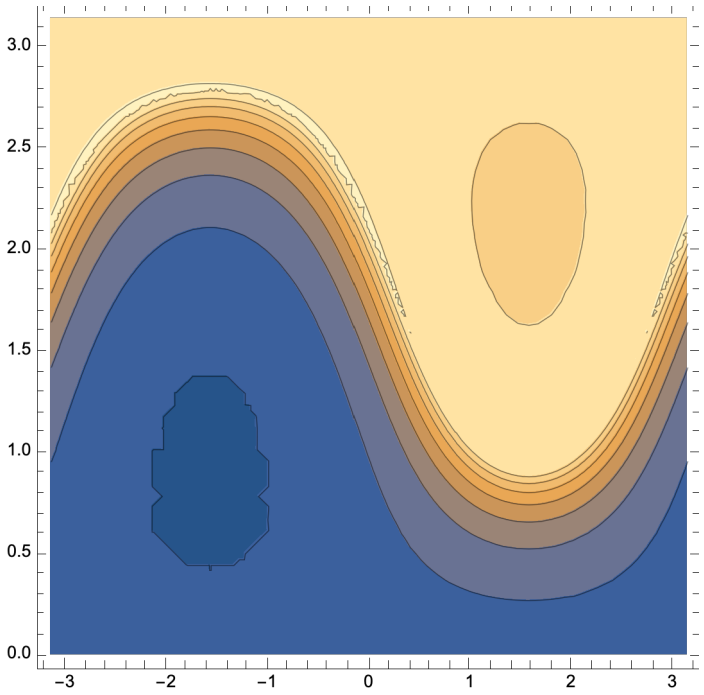
```
(InteratomicSurface[w, ncps, 1, {-3 π / 4, -π / 8}, BetaSphereRadii → beta0])[[2]]
```

```
2.84567
```

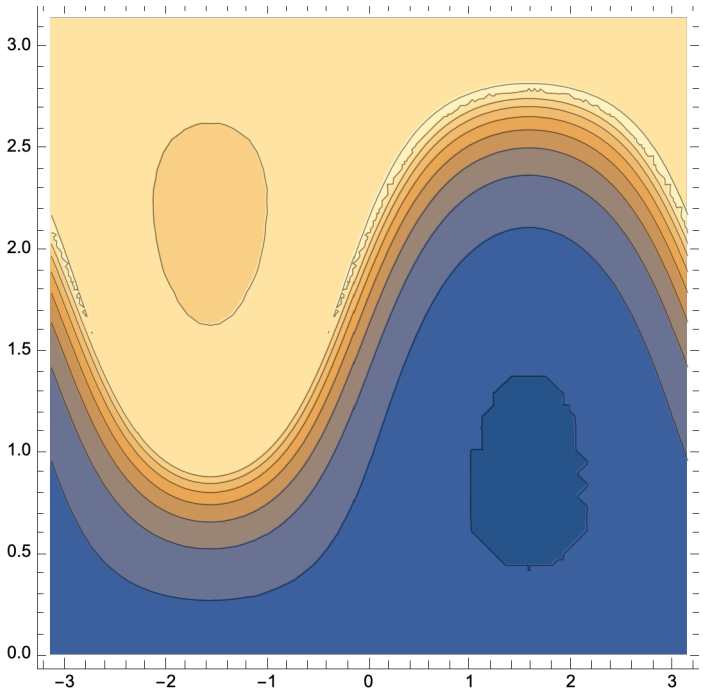
```
ias1ContourPlot = ContourPlot[
  (InteratomicSurface[w, ncps, 1, {theta, phi}, BetaSphereRadii → beta0])[[2]]
  , {phi, -π, π}
  , {theta, 0, π}
  , PlotRange → All]
```



```
ias2ContourPlot = ContourPlot[
  (InteratomicSurface[w, ncps, 2, {theta, phi}, BetaSphereRadii → beta0])[[2]]
  , {phi, -π, π}
  , {theta, 0, π}
  , PlotRange → All]
```



```
ias3ContourPlot = ContourPlot[
  (InteratomicSurface[w, ncps, 3, {theta, phi}, BetaSphereRadii → beta0])[[2]]
  , {phi, -π, π}
  , {theta, 0, π}
  , PlotRange → All]
```



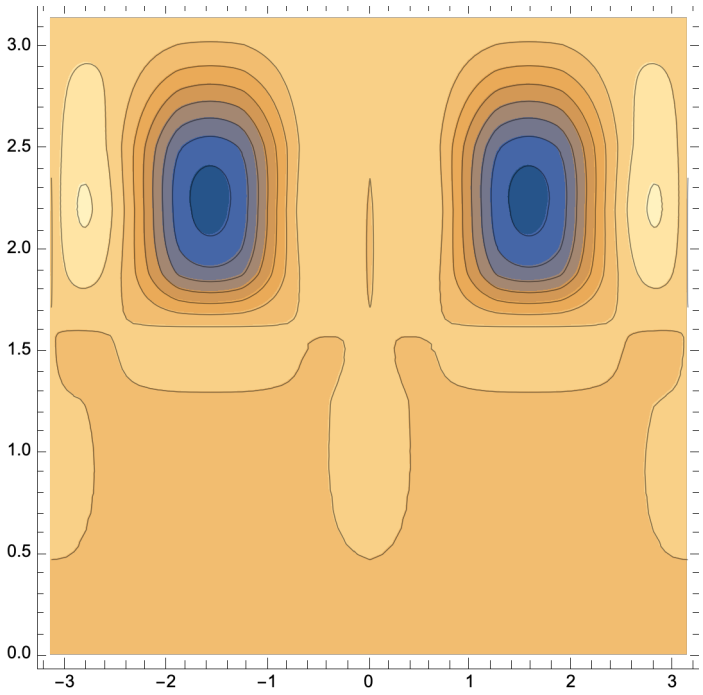
Hermite Polynomial Representation of rmax as a function of {θ, ϕ}

```
Options[FunctionInterpolation]
```

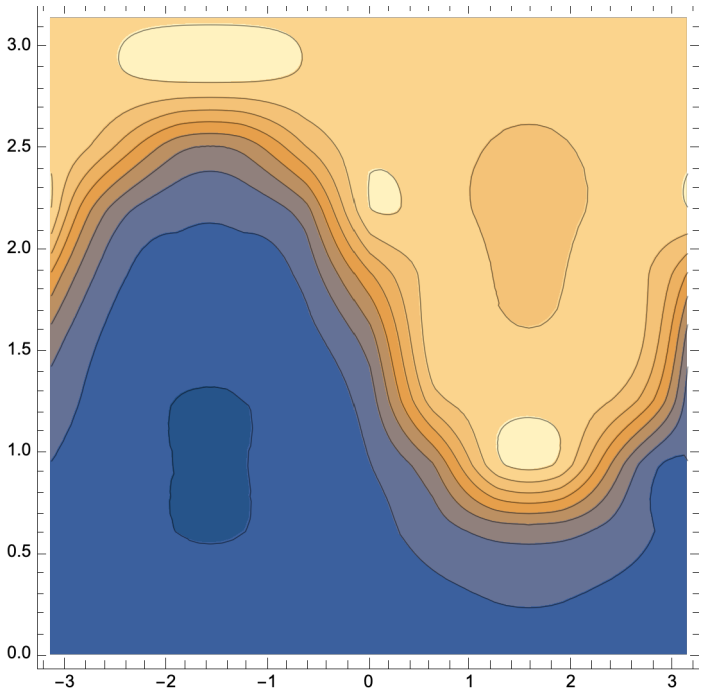
```
{AccuracyGoal → Automatic, InterpolationOrder → 3, InterpolationPoints → 11,
 InterpolationPrecision → Automatic, MaxRecursion → 6, PrecisionGoal → Automatic}
```

```
iasFunctionInterpolation = ParallelTable[
    FunctionInterpolation[
     (InteratomicSurface[w, ncps, i, {theta, phi}, BetaSphereRadii → beta0])[[2]]
     , {theta, N[0], N[π]}
     , {phi, N[-π], N[π]}
    ]
    , {i, 1, Length[ncps], 1}
    , Method → "FinestGrained"
  ];
```

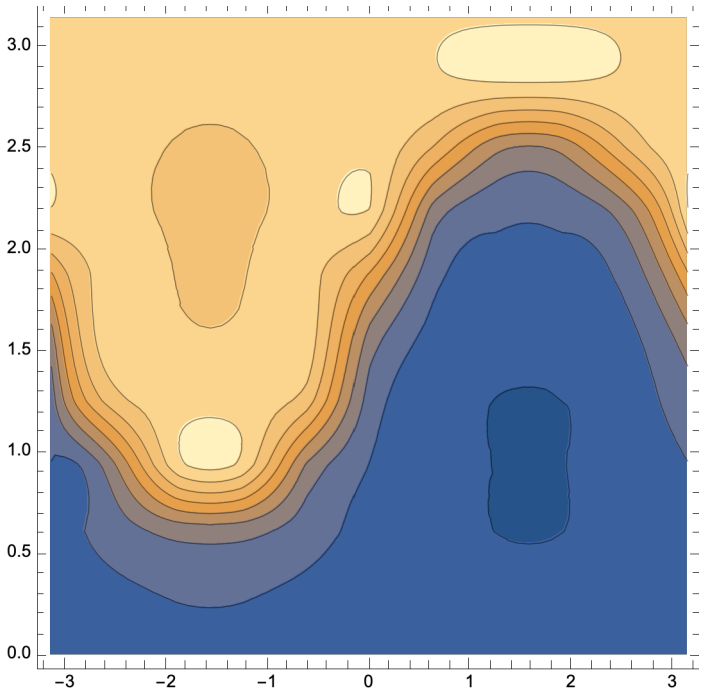
```
ContourPlot[
 (iasFunctionInterpolation[[1]])[theta, phi],
 {phi, -π, π},
 {theta, 0, π}
 , PlotRange → All]
```



```
ContourPlot[
 (iasFunctionInterpolation[[2]])[theta, phi],
 {phi, -π, π},
 {theta, 0, π}
 , PlotRange → All]
```



```
ContourPlot[
 (iasFunctionInterpolation[[3]])[theta, phi],
 {phi, -π, π},
 {theta, 0, π}
 , PlotRange → All]
```



Partitioning Radial and Angular Components

Split the integration into radial and angular components. Define a function that determines the radial limit and then integrates an arbitrary function **f**:

```
Yf[w_, ncps_, ncp_, {theta_?NumericQ, phi_?NumericQ}, f_] := Module[
    {rmax},
    rmax = (InteratomicSurface[w, ncps, ncp,
         {theta, phi}, BetaSphereRadii → beta0, MaxIterations → 10])[[2]];
    NIntegrate[
     r^2 f[w, {r Cos[phi] Sin[theta], r Sin[phi] Sin[theta], r Cos[theta]} + ncps[[ncp]]]
     , {r, N[0], N[rmax]}
     , Method → {"LocalAdaptive", "SymbolicProcessing" → 0}
    ]
  ];
Yfi[w_, ncps_, ncp_, {theta_?NumericQ, phi_?NumericQ}, f_] := Module[
    {rmax},
    rmax = (iasFunctionInterpolation[[ncp]])[theta, phi];
    NIntegrate[
     r^2 f[w, {r Cos[phi] Sin[theta], r Sin[phi] Sin[theta], r Cos[theta]} + ncps[[ncp]]]
     , {r, N[0], N[rmax]}
     , Method → {"LocalAdaptive", "SymbolicProcessing" → 0}
    ]
  ];
```

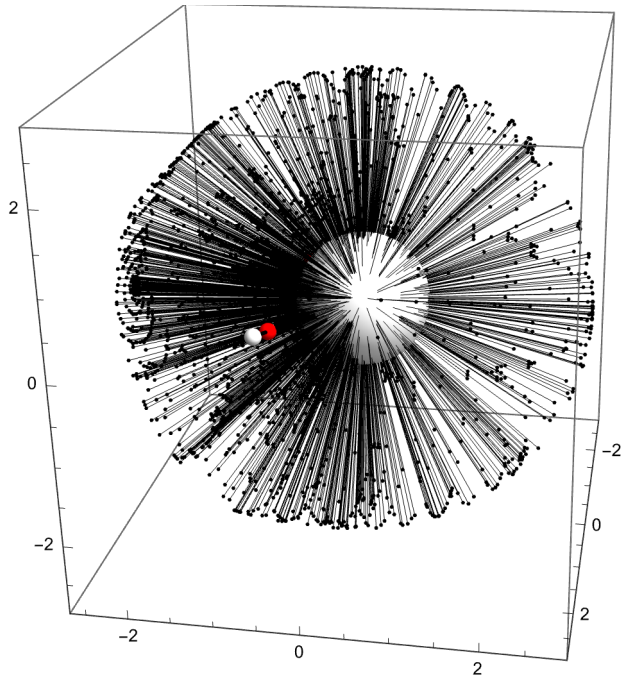
Atomic Surface

```
AbsoluteTiming[
 as1 = Reap[
    With[
     {ncp = 1},
     NIntegrate[
      Sin[theta] *
       Yf[w, ncps, ncp, {theta, phi}, rho]
      , {phi, N[-π], N[π]}
      , {theta, N[0], N[π]}
      , AccuracyGoal → 2, PrecisionGoal → Infinity
      , Method → {Automatic, "SymbolicProcessing" → 0}
      , EvaluationMonitor :→ Sow[(InteratomicSurface[
            w, ncps, ncp, {theta, phi}, BetaSphereRadii → beta0])[[1]]]
     ]
    ]
    ];
]
```

NIntegrate: Numerical integration converging too slowly; suspect one of the following: singularity, value of the integration is 0, highly oscillatory integrand, or WorkingPrecision too small.

```
{720.168, Null}
```

```
With[
 {ncp = 1},
 Show[
  ListPointPlot3D[as1[[2, 1]], ColorFunction → Function[{x, y, z}, Black]],
  Graphics3D[
   Table[
    Line[{ncps[[ncp]], as1[[2, 1, i]]}]
    , {i, 1, Length[as1[[2, 1]]]}]
   , Gray],
  graph,
  BoxRatios → Automatic
  , PlotRange → All]
]
```



Electron Density

```
(* Experimental
  f[theta_?NumericQ]:=Sin[theta]*NIntegrate[
   Yf[w,ncps,1,{theta,phi},rho]
   ,{phi,N[0],N[π]}
   ,Method→{"LocalAdaptive","SymbolicProcessing"→0}]
ParallelNInt[f,{N[-π],N[π]},2] *)
```

```
qO = AbsoluteTiming[
  With[{ncp = 1},
   NIntegrate[
    If[
     AssociatedNuclearAttractor[w, ncps, {x, y, z}, BetaSphereRadii → beta0] == ncp,
     rho[w, {x, y, z}],
     0],
    {x, bbxmin, ncps[[ncp, 1]], bbxmax},
    {y, bbymin, ncps[[ncp, 2]], bbymax},
    {z, bbzmin, ncps[[ncp, 3]], bbzmax},
    Method → {"LocalAdaptive", "SymbolicProcessing" → 0}
    , AccuracyGoal → 2, PrecisionGoal → Infinity]
  ]
 ]
```

```
{42.7339, 8.84525}
```

```
AbsoluteTiming[
 intq1 = With[
   {ncp = 1},
   NIntegrate[
    Sin[theta] *
     Yf[w, ncps, ncp, {theta, phi}, rho]
    , {phi, N[-π], N[π]}
    , {theta, N[0], N[π]}
    , AccuracyGoal → 2, PrecisionGoal → Infinity
    , Method → {"LocalAdaptive", "SymbolicProcessing" → 0}
   ]
  ]
]
```

```
{78.0846, 8.8292}
```

```
qO = AbsoluteTiming[
  With[{ncp = 1},
   NIntegrate[
    If[
     AssociatedNuclearAttractor[w, ncps, {x, y, z}, BetaSphereRadii → beta0] == ncp,
     rho[w, {x, y, z}],
     0],
    {x, bbxmin, ncps[[ncp, 1]], bbxmax},
    {y, bbymin, ncps[[ncp, 2]], bbymax},
    {z, bbzmin, ncps[[ncp, 3]], bbzmax},
    Method → {"LocalAdaptive", "SymbolicProcessing" → 0}
    , AccuracyGoal → 3, PrecisionGoal → Infinity]
  ]
 ]
```

```
{115.433, 8.83648}
```

```
AbsoluteTiming[
 intq1 = With[
   {ncp = 1},
   NIntegrate[
    Sin[theta] *
     Yf[w, ncps, ncp, {theta, phi}, rho]
    , {phi, N[-π], N[π]}
    , {theta, N[0], N[π]}
    , AccuracyGoal → 3, PrecisionGoal → Infinity
    , Method → {"LocalAdaptive", "SymbolicProcessing" → 0}
   ]
  ]
]
```

```
{255.15, 8.83347}
```

```
qO = AbsoluteTiming[
  With[{ncp = 1},
   NIntegrate[
    If[
     AssociatedNuclearAttractor[w, ncps, {x, y, z}, BetaSphereRadii → beta0] == ncp,
     rho[w, {x, y, z}],
     0],
    {x, bbxmin, ncps[[ncp, 1]], bbxmax},
    {y, bbymin, ncps[[ncp, 2]], bbymax},
    {z, bbzmin, ncps[[ncp, 3]], bbzmax},
    Method → {"LocalAdaptive", "SymbolicProcessing" → 0}
    , AccuracyGoal → 4, PrecisionGoal → Infinity]
  ]
 ]
```

```
{824.813, 8.83892}
```

```
AbsoluteTiming[
 intq1 = With[
   {ncp = 1},
   NIntegrate[
    Sin[theta] *
     Yf[w, ncps, ncp, {theta, phi}, rho]
    , {phi, N[-π], N[π]}
    , {theta, N[0], N[π]}
    , AccuracyGoal → 4, PrecisionGoal → Infinity
    , Method → {"LocalAdaptive", "SymbolicProcessing" → 0}
   ]
  ]
]
```

```
{1204.57, 8.83426}
```

```
AbsoluteTiming[
 intq2 = With[
   {ncp = 2},
   NIntegrate[
    Sin[theta] *
     Yf[w, ncps, ncp, {theta, phi}, rho]
    , {phi, N[-π], N[π]}
    , {theta, N[0], N[π]}
    , AccuracyGoal → 4, PrecisionGoal → Infinity
    , Method → {"LocalAdaptive", "SymbolicProcessing" → 0}
   ]
  ]
]
```

```
{647.004, 0.464007}
```

```
AbsoluteTiming[
 intq3 = With[
   {ncp = 3},
   NIntegrate[
    Sin[theta] *
     Yf[w, ncps, ncp, {theta, phi}, rho]
    , {phi, N[-π], N[π]}
    , {theta, N[0], N[π]}
    , AccuracyGoal → 4, PrecisionGoal → Infinity
    , Method → {"LocalAdaptive", "SymbolicProcessing" → 0}
   ]
  ]
]
```

```
{648.556, 0.464007}
```

```
intq1 + intq2 + intq3
```

```
9.76228
```

Laplacian of the Electron Density

```
AbsoluteTiming[
 intl1 = With[
   {ncp = 1},
   NIntegrate[
    Sin[theta] *
     Yf[w, ncps, ncp, {theta, phi}, laprho]
    , {phi, N[-π], N[π]}
    , {theta, N[0], N[π]}
    , AccuracyGoal → 4, PrecisionGoal → Infinity
    , Method → {"LocalAdaptive", "SymbolicProcessing" → 0}
   ]
  ]
]
```

```
AbsoluteTiming[
 intl2 = With[
   {ncp = 2},
   NIntegrate[
    Sin[theta] *
     Yf[w, ncps, ncp, {theta, phi}, laprho]
    , {phi, N[-π], N[π]}
    , {theta, N[0], N[π]}
    , AccuracyGoal → 4, PrecisionGoal → Infinity
    , Method → {"LocalAdaptive", "SymbolicProcessing" → 0}
   ]
  ]
]
```

```
AbsoluteTiming[
 intl3 = With[
   {ncp = 3},
   NIntegrate[
    Sin[theta] *
     Yf[w, ncps, ncp, {theta, phi}, laprho]
    , {phi, N[-π], N[π]}
    , {theta, N[0], N[π]}
    , AccuracyGoal → 4, PrecisionGoal → Infinity
    , Method → {"LocalAdaptive", "SymbolicProcessing" → 0}
   ]
  ]
]
```

```
intl1 + intl2 + intl3
```

Kinetic Energy Density G

```
AbsoluteTiming[
 intg1 = With[
   {ncp = 1},
   NIntegrate[
    Sin[theta] *
     Yf[w, ncps, ncp, {theta, phi}, KineticEnergyDensityG]
    , {phi, N[-π], N[π]}
    , {theta, N[0], N[π]}
    , AccuracyGoal → 4, PrecisionGoal → Infinity
    , Method → {"LocalAdaptive", "SymbolicProcessing" → 0}
   ]
  ]
]
```

```
AbsoluteTiming[
 intg2 = With[
   {ncp = 2},
   NIntegrate[
    Sin[theta] *
     Yf[w, ncps, ncp, {theta, phi}, KineticEnergyDensityG]
    , {phi, N[-π], N[π]}
    , {theta, N[0], N[π]}
    , AccuracyGoal → 4, PrecisionGoal → Infinity
    , Method → {"LocalAdaptive", "SymbolicProcessing" → 0}
   ]
  ]
]
```

```
AbsoluteTiming[
 intg3 = With[
   {ncp = 3},
   NIntegrate[
    Sin[theta] *
     Yf[w, ncps, ncp, {theta, phi}, KineticEnergyDensityG]
    , {phi, N[-π], N[π]}
    , {theta, N[0], N[π]}
    , AccuracyGoal → 4, PrecisionGoal → Infinity
    , Method → {"LocalAdaptive", "SymbolicProcessing" → 0}
   ]
  ]
]
```

```
intg1 + intg2 + intg3
```

Kinetic Energy Density K

```
AbsoluteTiming[
 intk1 = With[
   {ncp = 1},
   NIntegrate[
    Sin[theta] *
     Yf[w, ncps, ncp, {theta, phi}, KineticEnergyDensityK]
    , {phi, N[-π], N[π]}
    , {theta, N[0], N[π]}
    , AccuracyGoal → 4, PrecisionGoal → Infinity
    , Method → {"LocalAdaptive", "SymbolicProcessing" → 0}
   ]
  ]
]
```

```
AbsoluteTiming[
 intk2 = With[
   {ncp = 2},
   NIntegrate[
    Sin[theta] *
     Yf[w, ncps, ncp, {theta, phi}, KineticEnergyDensityK]
    , {phi, N[-π], N[π]}
    , {theta, N[0], N[π]}
    , AccuracyGoal → 4, PrecisionGoal → Infinity
    , Method → {"LocalAdaptive", "SymbolicProcessing" → 0}
   ]
  ]
]

AbsoluteTiming[
 intk3 = With[
   {ncp = 3},
   NIntegrate[
    Sin[theta] *
     Yf[w, ncps, ncp, {theta, phi}, KineticEnergyDensityK]
    , {phi, N[-π], N[π]}
    , {theta, N[0], N[π]}
    , AccuracyGoal → 4, PrecisionGoal → Infinity
    , Method → {"LocalAdaptive", "SymbolicProcessing" → 0}
   ]
  ]
]

intk1 + intk2 + intk3
```

Volume

```
One[x_, y_] := 1;
AbsoluteTiming[
 vint1 = With[
   {ncp = 1},
   NIntegrate[
    Sin[theta] *
     Yf[w, ncps, ncp, {theta, phi}, One]
    , {phi, N[-π], N[π]}
    , {theta, N[0], N[π]}
    , AccuracyGoal → 2, PrecisionGoal → Infinity
    , Method → {"LocalAdaptive", "SymbolicProcessing" → 0}
   ]
  ]
]
```

```
{3919.5, 73.6001}
```

```
One[x_, y_] := 1;
AbsoluteTiming[
 vint2 = With[
   {ncp = 2},
   NIntegrate[
    Sin[theta] *
     Yf[w, ncps, ncp, {theta, phi}, One]
    , {phi, N[-π], N[π]}
    , {theta, N[0], N[π]}
    , AccuracyGoal → 2, PrecisionGoal → Infinity
    , Method → {"LocalAdaptive", "SymbolicProcessing" → 0}
   ]
  ]
]
```

```
{1604.25, 12.1237}
```

```
One[x_, y_] := 1;
AbsoluteTiming[
 vint3 = With[
   {ncp = 3},
   NIntegrate[
    Sin[theta] *
     Yf[w, ncps, ncp, {theta, phi}, One]
    , {phi, N[-π], N[π]}
    , {theta, N[0], N[π]}
    , AccuracyGoal → 2, PrecisionGoal → Infinity
    , Method → {"LocalAdaptive", "SymbolicProcessing" → 0}
   ]
  ]
]
```

```
{1586.66, 12.1237}
```

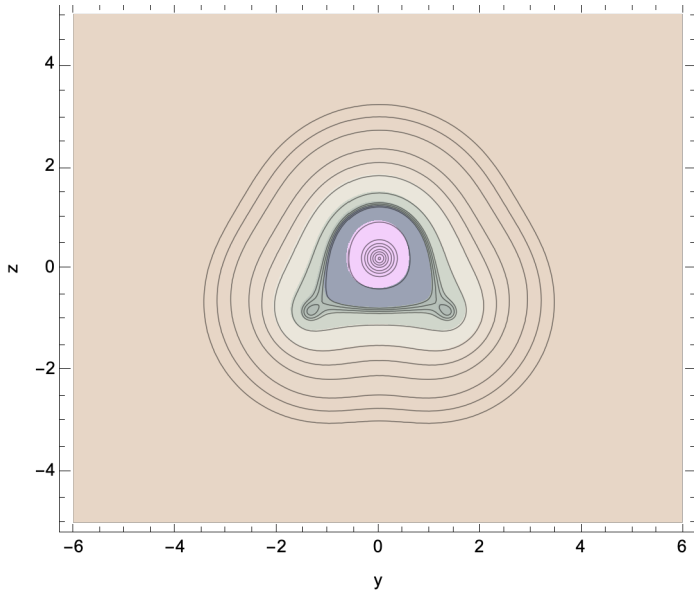
```
vint1 + vint2 + vint3
```

```
97.8475
```

Cerjan-Miller-Baker Approach to Locating Critical Points

The CMB method (introduced to QTAIM by Popelier) is amenable to direct location of stationary points of a particular signature in any field so long as we can provide its gradient and Hessian. In QTAIM we need separate routines for each corresponding to nuclear (-3), bond (-1), ring (+1), and cage (+3) critical points.

rhoPlot



```
CriticalPointGradientMinus3[
 {
  g[w, {0, 0, 1}],
  H[w, {0, 0, 1}]
 }]
```

$$\{3.23626 \times 10^{-19},\ 1.18208 \times 10^{-18},\ -1.81761\}$$

```
CriticalPointGradientMinus3[
 {
  g[w, {0, 0, -1}],
  H[w, {0, 0, -1}]
 }]
```

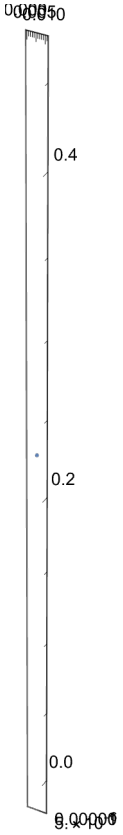
$$\{-8.35476 \times 10^{-18},\ -1.36815 \times 10^{-17},\ 2.23496\}$$

```
CriticalPointRank[{x_, y_, z_}] :=
  Total[Map[If[# == 0, 0, 1] &, Chop[Eigenvalues[H[w, {x, y, z}]]]]];
CriticalPointSignature[{x_, y_, z_}] :=
  Total[Map[Which[# < 0, -1, # == 0, 0, # > 0, 1] &, Chop[Eigenvalues[H[w, {x, y, z}]]]]];
```

```
AbsoluteTiming[
 soln = NDSolveValue[
   {
    X'[s] == CriticalPointGradientMinus3[
      {
       g[w, X[s]],
       H[w, X[s]]
      }],
    X[0] == {ncps⟦1, 1⟧ + 0.005, ncps⟦1, 2⟧ + 0.005, ncps⟦1, 3⟧ - 0.005}
   },
   {X[s]},
   {s, 0, 10 000}
   (*,Method→"BDF"*)
   (*,AccuracyGoal→2,PrecisionGoal→Infinity]*)
  ]
]
```

$$\{25.2718,\ \{\text{InterpolatingFunction}[\ \langle\text{Domain: } \{\{0.,\ 1.00\times10^{4}\}\}\ \text{Output dimensions: } \{3\}\rangle\ ][s]\}\}$$

```
ParametricPlot3D[
 soln[[1]]
 , {s, 0, soln[[1, 0, 1, 1, 2]]}, PlotRange → All]
```

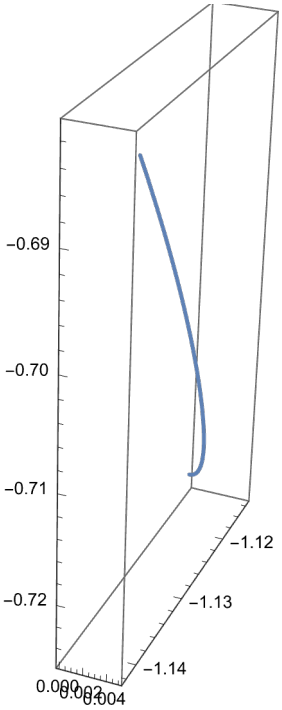



```
soln = NDSolveValue[
  {
   X'[s] == CriticalPointGradientMinus1[
     {
      g[w, X[s]],
      H[w, X[s]]
     }],
   X[0] == {bcps[[1, 1]] + 0.005, bcps[[1, 2]] + 0.005, bcps[[1, 3]] - 0.005}
  },
  {X[s]},
  {s, 0, 10 000}
  (*,Method→"BDF"
  ,AccuracyGoal→2,PrecisionGoal→Infinity*)]
```

```
soln[[1]]
```

InterpolatingFunction[Domain: {{0., 0.00155}} Output dimensions: {3}][s]

```
ParametricPlot3D[
 soln[[1]]
 , {s, 0, soln[[1, 0, 1, 1, 2]]}, PlotRange → All]
```




```
sf = soln[[1, 0, 1, 1, 2]]
```

1.47724×10^{-7}

```
First[soln /. {s → soln[[1, 0, 1, 1, 2]]}]
```

{0.0055319, 0.00001919, 0.225367}

```
FindRoot[
 CriticalPointGradientMinus1[
  {
   g[w, {x, y, z}],
   H[w, {x, y, z}]
  }]
 , {x, bcps[[1, 1]] + 0.005}, {y, bcps[[1, 2]] + 0.005}, {z, bcps[[1, 3]] - 0.005}
 , EvaluationMonitor :→ Print[{x, y, z}]
]
```

{0.005, −1.14144, −0.682648}

{0.005, −1.14144, −0.682648}

{0.00500001, −1.14144, −0.682648}

{0.005, −1.14144, −0.682648}

{0.005, −1.14144, −0.682648}

```
{0.00584397, -1.05317, -0.624054}
{0.00584397, -1.05317, -0.624054}
{0.00584398, -1.05317, -0.624054}
{0.00584397, -1.05317, -0.624054}
{0.00584397, -1.05317, -0.624054}
{0.00486289, -0.974382, -0.580907}
{0.00486289, -0.974382, -0.580907}
{0.0048629, -0.974382, -0.580907}
{0.00486289, -0.974382, -0.580907}
{0.00486289, -0.974382, -0.580907}
{0.00219441, -0.901026, -0.559582}
{0.00219441, -0.901026, -0.559582}
{0.00219443, -0.901026, -0.559582}
{0.00219441, -0.901026, -0.559582}
{0.00219441, -0.901026, -0.559582}
{-0.0000391602, -0.82884, -0.572998}
{-0.0000391602, -0.82884, -0.572998}
{-0.0000391453, -0.82884, -0.572998}
{-0.0000391602, -0.82884, -0.572998}
{-0.0000391602, -0.82884, -0.572998}
{0.0000522522, -0.748024, -0.640551}
{-0.0000300189, -0.820759, -0.579754}
{-0.0000300189, -0.820759, -0.579754}
{-0.000030004, -0.820759, -0.579754}
{-0.0000300189, -0.820759, -0.579754}
{-0.0000300189, -0.820759, -0.579754}
{0.0000571668, -0.735833, -0.654928}
{-0.0000213004, -0.812266, -0.587271}
{-0.0000213004, -0.812266, -0.587271}
{-0.0000212855, -0.812266, -0.587271}
{-0.0000213004, -0.812266, -0.587271}
{-0.0000213004, -0.812266, -0.587271}
{0.0000617959, -0.722893, -0.670323}
{-0.0000129907, -0.803329, -0.595576}
{-0.0000129907, -0.803329, -0.595576}
{-0.0000129758, -0.803329, -0.595576}
```

{-0.0000129907, -0.803329, -0.595576}

{-0.0000129907, -0.803329, -0.595576}

{0.0000657492, -0.709165, -0.686706}

$\{-5.11674 \times 10^{-6}, -0.793913, -0.604689\}$

$\{-5.11674 \times 10^{-6}, -0.793913, -0.604689\}$

$\{-5.10184 \times 10^{-6}, -0.793913, -0.604689\}$

$\{-5.11674 \times 10^{-6}, -0.793913, -0.604689\}$

$\{-5.11674 \times 10^{-6}, -0.793913, -0.604689\}$

{0.0000670345, -0.694611, -0.704031}

$\{2.09838 \times 10^{-6}, -0.783982, -0.614623\}$

$\{-1.50918 \times 10^{-6}, -0.788947, -0.609656\}$

$\{-1.50918 \times 10^{-6}, -0.788947, -0.609656\}$

$\{-1.49428 \times 10^{-6}, -0.788947, -0.609656\}$

$\{-1.50918 \times 10^{-6}, -0.788947, -0.609656\}$

$\{-1.50918 \times 10^{-6}, -0.788947, -0.609656\}$

{0.0000664816, -0.686883, -0.713197}

$\{5.28989 \times 10^{-6}, -0.778741, -0.62001\}$

$\{1.89036 \times 10^{-6}, -0.783844, -0.614833\}$

$\{1.90587 \times 10^{-7}, -0.786396, -0.612245\}$

$\{-6.59298 \times 10^{-7}, -0.787672, -0.61095\}$

$\{-6.59298 \times 10^{-7}, -0.787672, -0.61095\}$

$\{-6.44396 \times 10^{-7}, -0.787672, -0.61095\}$

$\{-6.59298 \times 10^{-7}, -0.787672, -0.61095\}$

$\{-6.59298 \times 10^{-7}, -0.787672, -0.61095\}$

{0.0000667433, -0.68489, -0.715555}

$\{6.08096 \times 10^{-6}, -0.777394, -0.621411\}$

$\{2.71083 \times 10^{-6}, -0.782533, -0.616181\}$

$\{1.02577 \times 10^{-6}, -0.785102, -0.613566\}$

$\{1.83235 \times 10^{-7}, -0.786387, -0.612258\}$

$\{-2.38031 \times 10^{-7}, -0.787029, -0.611604\}$

$\{-2.38031 \times 10^{-7}, -0.787029, -0.611604\}$

$\{-2.2313 \times 10^{-7}, -0.787029, -0.611604\}$

$\{-2.38031 \times 10^{-7}, -0.787029, -0.611604\}$

$\{-2.38031 \times 10^{-7}, -0.787029, -0.611604\}$

$$\{0.000066788, -0.683887, -0.716742\}$$

$$\{6.46457 \times 10^{-6}, -0.776715, -0.622118\}$$

$$\{3.11327 \times 10^{-6}, -0.781872, -0.616861\}$$

$$\{1.43762 \times 10^{-6}, -0.784451, -0.614233\}$$

$$\{5.99794 \times 10^{-7}, -0.78574, -0.612918\}$$

$$\{1.80881 \times 10^{-7}, -0.786385, -0.612261\}$$

$$\{-2.85749 \times 10^{-8}, -0.786707, -0.611933\}$$

$$\{-2.85749 \times 10^{-8}, -0.786707, -0.611933\}$$

$$\{-1.36737 \times 10^{-8}, -0.786707, -0.611933\}$$

$$\{-2.85749 \times 10^{-8}, -0.786707, -0.611933\}$$

$$\{-2.85749 \times 10^{-8}, -0.786707, -0.611933\}$$

$$\{0.0000663097, -0.683383, -0.717338\}$$

$$\{6.60525 \times 10^{-6}, -0.776375, -0.622473\}$$

$$\{3.28834 \times 10^{-6}, -0.781541, -0.617203\}$$

$$\{1.62988 \times 10^{-6}, -0.784124, -0.614568\}$$

$$\{8.00653 \times 10^{-7}, -0.785415, -0.61325\}$$

$$\{3.86039 \times 10^{-7}, -0.786061, -0.612592\}$$

$$\{1.78732 \times 10^{-7}, -0.786384, -0.612262\}$$

$$\{7.50786 \times 10^{-8}, -0.786545, -0.612097\}$$

$$\{2.32519 \times 10^{-8}, -0.786626, -0.612015\}$$

$$\{-2.6615 \times 10^{-9}, -0.786667, -0.611974\}$$

$$\{-2.6615 \times 10^{-9}, -0.786667, -0.611974\}$$

$$\{1.22397 \times 10^{-8}, -0.786667, -0.611974\}$$

$$\{-2.6615 \times 10^{-9}, -0.786667, -0.611974\}$$

$$\{-2.6615 \times 10^{-9}, -0.786667, -0.611974\}$$

$$\{0.000066223, -0.68332, -0.717412\}$$

$$\{6.61991 \times 10^{-6}, -0.776332, -0.622518\}$$

$$\{3.30862 \times 10^{-6}, -0.781499, -0.617246\}$$

$$\{1.65298 \times 10^{-6}, -0.784083, -0.61461\}$$

$$\{8.2516 \times 10^{-7}, -0.785375, -0.613292\}$$

$$\{4.11249 \times 10^{-7}, -0.786021, -0.612633\}$$

$$\{2.04294 \times 10^{-7}, -0.786344, -0.612303\}$$

$$\{1.00816 \times 10^{-7}, -0.786505, -0.612139\}$$

$$\{4.90773 \times 10^{-8}, -0.786586, -0.612056\}$$

$\{2.32079 \times 10^{-8}, -0.786626, -0.612015\}$

$\{1.02732 \times 10^{-8}, -0.786646, -0.611995\}$

$\{3.80585 \times 10^{-9}, -0.786656, -0.611984\}$

$\{5.72175 \times 10^{-10}, -0.786662, -0.611979\}$

$\{-1.04466 \times 10^{-9}, -0.786664, -0.611977\}$

$\{-1.04466 \times 10^{-9}, -0.786664, -0.611977\}$

$\{1.38565 \times 10^{-8}, -0.786664, -0.611977\}$

$\{-1.04466 \times 10^{-9}, -0.786664, -0.611977\}$

$\{-1.04466 \times 10^{-9}, -0.786664, -0.611977\}$

$\{0.0000662244, -0.683316, -0.717417\}$

$\{6.6215 \times 10^{-6}, -0.776329, -0.622521\}$

$\{3.31023 \times 10^{-6}, -0.781497, -0.617249\}$

$\{1.65459 \times 10^{-6}, -0.78408, -0.614613\}$

$\{8.26773 \times 10^{-7}, -0.785372, -0.613295\}$

$\{4.12864 \times 10^{-7}, -0.786018, -0.612636\}$

$\{2.0591 \times 10^{-7}, -0.786341, -0.612306\}$

$\{1.02433 \times 10^{-7}, -0.786503, -0.612141\}$

$\{5.0694 \times 10^{-8}, -0.786583, -0.612059\}$

$\{2.48246 \times 10^{-8}, -0.786624, -0.612018\}$

$\{1.189 \times 10^{-8}, -0.786644, -0.611997\}$

$\{5.42266 \times 10^{-9}, -0.786654, -0.611987\}$

$\{2.189 \times 10^{-9}, -0.786659, -0.611982\}$

$\{5.72168 \times 10^{-10}, -0.786662, -0.611979\}$

$\{-2.36248 \times 10^{-10}, -0.786663, -0.611978\}$

$\{-2.36248 \times 10^{-10}, -0.786663, -0.611978\}$

$\{1.46649 \times 10^{-8}, -0.786663, -0.611978\}$

$\{-2.36248 \times 10^{-10}, -0.786663, -0.611978\}$

$\{-2.36248 \times 10^{-10}, -0.786663, -0.611978\}$

$\{0.0000662486, -0.683314, -0.71742\}$

$\{6.62465 \times 10^{-6}, -0.776328, -0.622522\}$

$\{3.31221 \times 10^{-6}, -0.781495, -0.61725\}$

$\{1.65598 \times 10^{-6}, -0.784079, -0.614614\}$

$\{8.27874 \times 10^{-7}, -0.785371, -0.613296\}$

$\{4.13819 \times 10^{-7}, -0.786017, -0.612637\}$

$\{2.06791 \times 10^{-7}, -0.78634, -0.612307\}$

$\{1.03278 \times 10^{-7}, -0.786501, -0.612143\}$

$\{5.15207 \times 10^{-8}, -0.786582, -0.61206\}$

$\{2.56422 \times 10^{-8}, -0.786622, -0.612019\}$

$\{1.2703 \times 10^{-8}, -0.786643, -0.611998\}$

$\{6.23337 \times 10^{-9}, -0.786653, -0.611988\}$

$\{2.99856 \times 10^{-9}, -0.786658, -0.611983\}$

$\{1.38116 \times 10^{-9}, -0.78666, -0.61198\}$

$\{5.72454 \times 10^{-10}, -0.786662, -0.611979\}$

$\{1.68103 \times 10^{-10}, -0.786662, -0.611978\}$

$\{-3.40723 \times 10^{-11}, -0.786662, -0.611978\}$

$\{-3.40723 \times 10^{-11}, -0.786662, -0.611978\}$

$\{1.48671 \times 10^{-8}, -0.786662, -0.611978\}$

$\{-3.40723 \times 10^{-11}, -0.786662, -0.611978\}$

$\{-3.40723 \times 10^{-11}, -0.786662, -0.611978\}$

$\{0.0000678417, -0.683312, -0.717426\}$

$\{6.78414 \times 10^{-6}, -0.776327, -0.622523\}$

$\{3.39205 \times 10^{-6}, -0.781495, -0.617251\}$

$\{1.69601 \times 10^{-6}, -0.784079, -0.614614\}$

$\{8.47988 \times 10^{-7}, -0.785371, -0.613296\}$

$\{4.23977 \times 10^{-7}, -0.786017, -0.612637\}$

$\{2.11971 \times 10^{-7}, -0.78634, -0.612308\}$

$\{1.05969 \times 10^{-7}, -0.786501, -0.612143\}$

$\{5.29673 \times 10^{-8}, -0.786582, -0.612061\}$

$\{2.64666 \times 10^{-8}, -0.786622, -0.612019\}$

$\{1.32163 \times 10^{-8}, -0.786642, -0.611999\}$

$\{6.5911 \times 10^{-9}, -0.786652, -0.611988\}$

$\{3.27851 \times 10^{-9}, -0.786657, -0.611983\}$

$\{1.62222 \times 10^{-9}, -0.78666, -0.611981\}$

$\{7.94074 \times 10^{-10}, -0.786661, -0.611979\}$

$\{3.80001 \times 10^{-10}, -0.786662, -0.611979\}$

$\{1.72964 \times 10^{-10}, -0.786662, -0.611978\}$

$\{6.94461 \times 10^{-11}, -0.786662, -0.611978\}$

$\{1.76869 \times 10^{-11}, -0.786662, -0.611978\}$

$\{-8.19267 \times 10^{-12}, -0.786662, -0.611978\}$

$\{-8.19267 \times 10^{-12}, -0.786662, -0.611978\}$

$\{1.4893 \times 10^{-8}, -0.786662, -0.611978\}$

$\{-8.19267 \times 10^{-12}, -0.786662, -0.611978\}$

$\{-8.19267 \times 10^{-12}, -0.786662, -0.611978\}$

$\{0.000152335, -0.683294, -0.717493\}$

$\{0.0000152335, -0.776326, -0.62253\}$

$\{7.61673 \times 10^{-6}, -0.781494, -0.617254\}$

$\{3.80836 \times 10^{-6}, -0.784078, -0.614616\}$

$\{1.90418 \times 10^{-6}, -0.78537, -0.613297\}$

$\{9.52085 \times 10^{-7}, -0.786016, -0.612638\}$

$\{4.76038 \times 10^{-7}, -0.786339, -0.612308\}$

$\{2.38015 \times 10^{-7}, -0.786501, -0.612143\}$

$\{1.19003 \times 10^{-7}, -0.786582, -0.612061\}$

$\{5.94976 \times 10^{-8}, -0.786622, -0.612019\}$

$\{2.97447 \times 10^{-8}, -0.786642, -0.611999\}$

$\{1.48683 \times 10^{-8}, -0.786652, -0.611988\}$

$\{7.43003 \times 10^{-9}, -0.786657, -0.611983\}$

$\{3.71092 \times 10^{-9}, -0.78666, -0.611981\}$

$\{1.85136 \times 10^{-9}, -0.786661, -0.611979\}$

$\{9.21586 \times 10^{-10}, -0.786662, -0.611979\}$

$\{4.56696 \times 10^{-10}, -0.786662, -0.611978\}$

$\{2.24252 \times 10^{-10}, -0.786662, -0.611978\}$

$\{1.0803 \times 10^{-10}, -0.786662, -0.611978\}$

$\{4.99185 \times 10^{-11}, -0.786662, -0.611978\}$

$\{2.08629 \times 10^{-11}, -0.786662, -0.611978\}$

$\{6.33511 \times 10^{-12}, -0.786662, -0.611978\}$

$\{-9.28778 \times 10^{-13}, -0.786662, -0.611978\}$

$\{-9.28778 \times 10^{-13}, -0.786662, -0.611978\}$

$\{1.49002 \times 10^{-8}, -0.786662, -0.611978\}$

••• Power: Infinite expression $\frac{1}{0.}$ encountered.

••• Power: Infinite expression $\frac{1}{0.}$ encountered.

Power: Infinite expression $\frac{1}{0.}$ encountered.

General: Further output of Power::infy will be suppressed during this calculation.

FindRoot: The function value {ComplexInfinity, ComplexInfinity, ComplexInfinity} is not a list of numbers with dimensions {3} at {x, y, z} = $\{-9.28778\times10^{-13}, -0.786662, -0.611978\}$.

$\{x \to -9.28778\times10^{-13}, y \to -0.786662, z \to -0.611978\}$

```
bcps // TableForm
```

```
0    -1.14644    -0.677648    3    -1
0    1.14644     -0.677648    3    -1
```

```
(*
FindRoot[
 CriticalPointGradientPlus1[
  {
   g[w,{x,y,z}],
   H[w,{x,y,z}]
  }]
 ,{x,rcps[[1,1]]+0.005},{y,rcps[[1,2]]+0.005},{z,rcps[[1,3]]-0.005}
 ,EvaluationMonitor:>Print[{x,y,z}]
]
*)
```

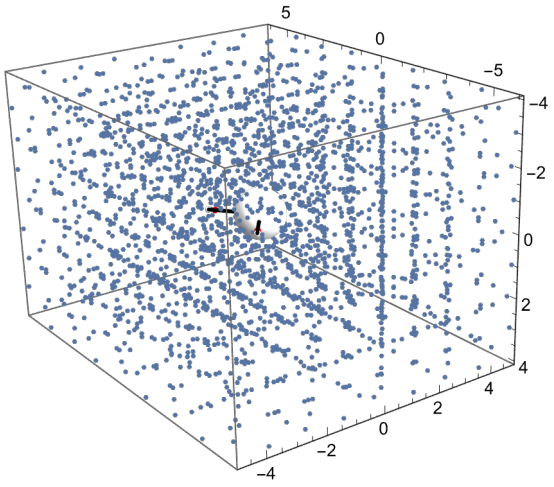
```
(*
FindRoot[
 CriticalPointGradientPlus3[
  {
   g[w,{x,y,z}],
   H[w,{x,y,z}]
  }]
 ,{x,ccps[[1,1]]+0.005},{y,ccps[[1,2]]+0.005},{z,ccps[[1,3]]-0.005}
 ,EvaluationMonitor:>Print[{x,y,z}]
]
*)
```

Parallel Search

A strategy is to make a grid and search for each type of critical point. Let's start with the integration of the electron density over the system:

```
q = Reap[
    AbsoluteTiming[
     NIntegrate[
      If[rho[w, {x, y, z}] > Min[$QTAIMContoursPositive],
       rho[w, {x, y, z}],
       0
      ],
      {x, bbxmin, bbxmax},
      {y, bbymin, bbymax},
      {z, bbzmin, bbzmax},
      Method → {"LocalAdaptive", "SymbolicProcessing" → 0}
      , AccuracyGoal → 0, PrecisionGoal → Infinity
      , EvaluationMonitor :→ Sow[{x, y, z}]]
    ]
   ];
integrationPoints = q[[2, 1]];
```

```
Show[
 ListPointPlot3D[integrationPoints, AspectRatio → Automatic],
 graph
 , BoxRatios → Automatic
 , PlotRange → All
]
```



But this is probably way too many points. We would reserve this strategy for careful scans of systems of which we knew very little, and are looking for subtle/long-range bond paths, etc.

Example: Offset slightly from the first bond critical point, and then perform CJM gradient ascent to a -1 critical point:

```
soln = NDSolveValue[
  {
   X'[s] == CriticalPointGradientMinus1[
     {
      g[w, X[s]],
      H[w, X[s]]
     }],
   X[0] == {bcps[[1, 1]] + 0.005, bcps[[1, 2]] + 0.005, bcps[[1, 3]] - 0.005}
  },
  {X[s]},
  {s, 0, 10 000}
  , Method → "BDF"
  , MaxSteps → 100
 (*,AccuracyGoal→2,PrecisionGoal→Infinity*)]
```

NDSolveValue: Maximum number of 100 steps reached at the point s == 0.0015268759714776791`.

{InterpolatingFunction[Domain: {{0., 0.00153}} Output dimensions: {3}][s]}

(The endpoint is buried in the interpolating function object result)

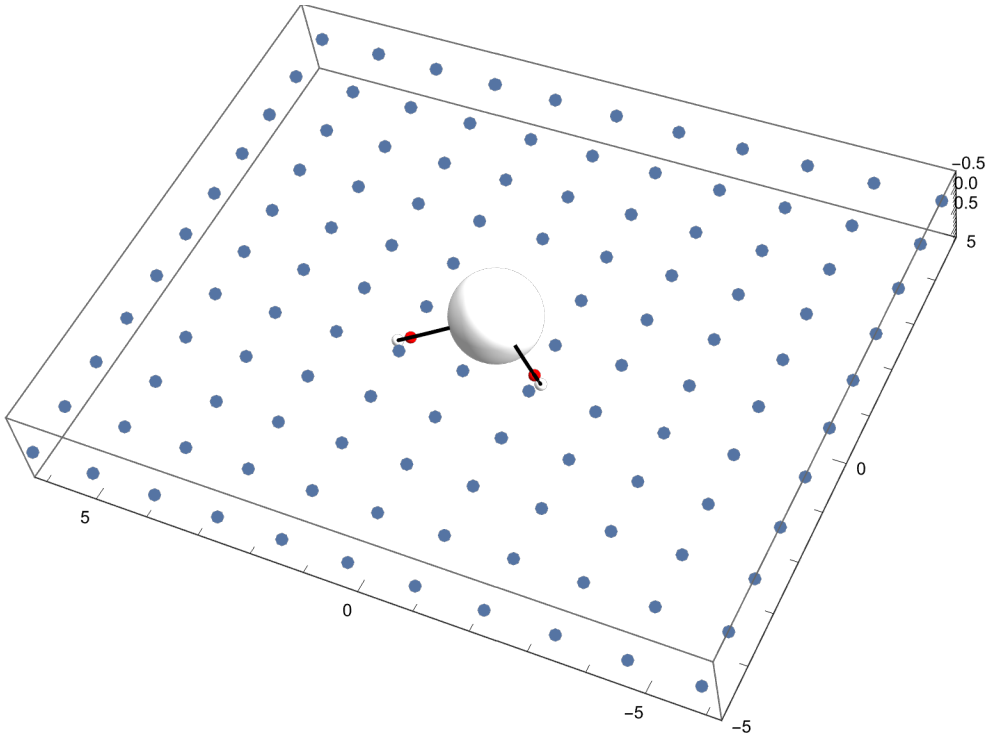
This search starts at each nuclear position and performs a CJM gradient ascent:

```
AbsoluteTiming[
  cpsMinus3 = Select[
    DeleteDuplicates[
     Select[
      Select[
       ParallelTable[
        Quiet[Check[
          soln = NDSolveValue[
            {
             X'[s] == CriticalPointGradientMinus3[
               {
                g[w, X[s]],
                H[w, X[s]]
               }],
             X[0] == (w["NuclearCartesianCoordinates"])[[i]]
            },
            {X[s]},
            {s, 0, 10 000}
            , Method → "BDF"
            , MaxSteps → 100
            (*,AccuracyGoal→4,PrecisionGoal→Infinity*)];
          First[soln /. {s → soln[[1, 0, 1, 1, 2]]}]
          ,
          {}
         ]]
        ,
        {i, 1, w["NumberOfNuclei"], 1}
        , Method → "FinestGrained"
       ]
       , Length[#] == 3 &]
      , ((bbxmin ≤ #[[1]] ≤ bbxmax) &&
          (bbymin ≤ #[[2]] ≤ bbymax) &&
          (bbzmin ≤ #[[3]] ≤ bbzmax)) &]
     , (EuclideanDistance[#1, #2] < 1*^-3) &]
    , CriticalPointCharacter[#] == 3 && CriticalPointSignature[#] == -3 &
   ]
 ];

cpsMinus3 // Chop // TableForm
```

```
{}
```

A table of starting values in the *yz* plane:

```
initialValues = Flatten[
   Outer[
    List,
    {0.},
    Subdivide[bbymin, bbymax, 10],
    Subdivide[bbzmin, bbzmax, 10]
   ]
   , 2];

Show[
 ListPointPlot3D[initialValues, AspectRatio → Automatic],
 graph
 , BoxRatios → Automatic
 , PlotRange → All
]
```



```
AbsoluteTiming[
 cpsMinus3 = Select[
    DeleteDuplicates[
     Select[
      Select[
       ParallelTable[
        Quiet[Check[
          soln = NDSolveValue[
            {
             X'[s] == CriticalPointGradientMinus3[
               {
                g[w, X[s]],
                H[w, X[s]]
               }],
             X[0] == initialValues[[i]]
            },
            {X[s]},
            {s, 0, 10 000}
            , Method → "BDF"
            , MaxSteps → 100
            (*,AccuracyGoal→4,PrecisionGoal→Infinity*)];
          First[soln /. {s → soln[[1, 0, 1, 1, 2]]}]
          ,
          {}
         ]]
        ,
        {i, 1, Length[initialValues], 1}
        , Method → "FinestGrained"
       ]
       , Length[#] == 3 &]
      , ((bbxmin ≤ #[[1]] ≤ bbxmax) &&
          (bbymin ≤ #[[2]] ≤ bbymax) &&
          (bbzmin ≤ #[[3]] ≤ bbzmax)) &]
     , (EuclideanDistance[#1, #2] < 1*^-3) &]
    , CriticalPointCharacter[#] == 3 && CriticalPointSignature[#] == -3 &
   ];
]
```

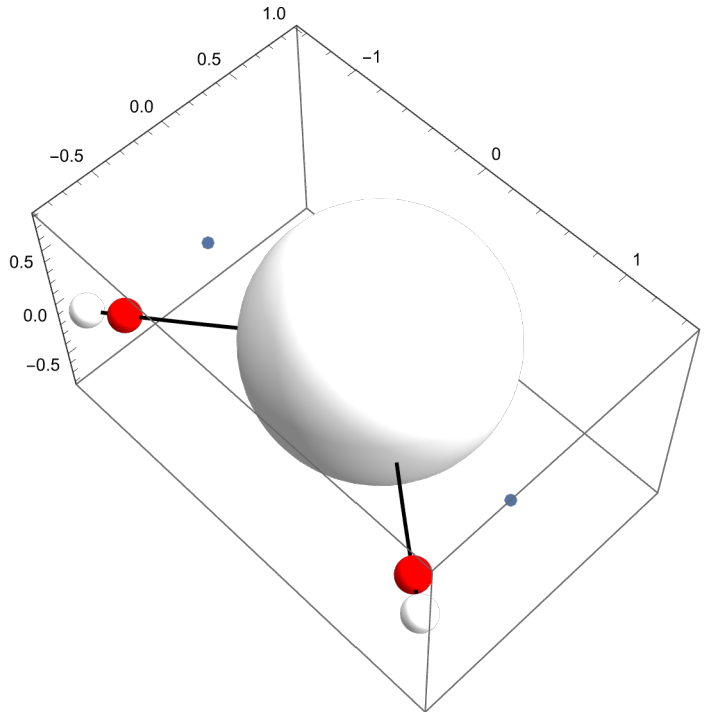
```
{1.38897, Null}
```

```
cpsMinus3 // Chop // TableForm
```

```
{}
```

Initial values on the same grid, provided they are enclosed by min/max of NCP coordinates (heuristic):

```
initialValues = Select[
   Flatten[
    Outer[
     List,
     {0.},
     Subdivide[bbymin, bbymax, 10],
     Subdivide[bbzmin, bbzmax, 10]
    ]
    , 2],
   (Min[ncps[[All, 1]]] ≤ #[[1]] ≤ Max[ncps[[All, 1]]] &&
      Min[ncps[[All, 2]]] ≤ #[[2]] ≤ Max[ncps[[All, 2]]] &&
      Min[ncps[[All, 3]]] ≤ #[[3]] ≤ Max[ncps[[All, 3]]])
    &];

Show[
 ListPointPlot3D[initialValues, AspectRatio → Automatic],
 graph
 , BoxRatios → Automatic
 , PlotRange → All
]
```



```
AbsoluteTiming[
 cpsMinus1 = Select[
    DeleteDuplicates[
     Select[
      Select[
       ParallelTable[
        Quiet[Check[
          soln = NDSolveValue[
            {
             X'[s] == CriticalPointGradientMinus1[
               {
                g[w, X[s]],
                H[w, X[s]]
               }],
             X[0] == initialValues[[i]]
            },
            {X[s]},
            {s, 0, 10 000}
            , Method → "BDF"
            , MaxSteps → 100
            (*,AccuracyGoal→4,PrecisionGoal→Infinity*)];
          First[soln /. {s → soln[[1, 0, 1, 1, 2]]}]
          ,
          {}
         ]]
        ,
        {i, 1, Length[initialValues], 1}
        , Method → "FinestGrained"
       ]
       , Length[#] == 3 &]
      , ((bbxmin ≤ #[[1]] ≤ bbxmax) &&
          (bbymin ≤ #[[2]] ≤ bbymax) &&
          (bbzmin ≤ #[[3]] ≤ bbzmax)) &]
     , (EuclideanDistance[#1, #2] < 1*^-3) &]
    , CriticalPointCharacter[#] == 3 && CriticalPointSignature[#] == -1 &
   ];
]
```

```
{0.071329, Null}
```

```
cpsMinus1 // Chop // TableForm
```

```
{}
```

```
AbsoluteTiming[
 cpsPlus1 = Select[
    DeleteDuplicates[
     Select[
      Select[
       ParallelTable[
        Quiet[Check[
          soln = NDSolveValue[
            {
             X'[s] == CriticalPointGradientPlus1[
               {
                g[w, X[s]],
                H[w, X[s]]
               }],
             X[0] == initialValues[[i]]
            },
            {X[s]},
            {s, 0, 10 000}
            , Method → "BDF"
            , MaxSteps → 100
            (*,AccuracyGoal→4,PrecisionGoal→Infinity*)];
          First[soln /. {s → soln[[1, 0, 1, 1, 2]]}]
          ,
          {}
         ]]
        ,
        {i, 1, Length[initialValues], 1}
        , Method → "FinestGrained"
       ]
       , Length[#] == 3 &]
      , ((bbxmin ≤ #[[1]] ≤ bbxmax) &&
          (bbymin ≤ #[[2]] ≤ bbymax) &&
          (bbzmin ≤ #[[3]] ≤ bbzmax)) &]
     , (EuclideanDistance[#1, #2] < 1*^-3) &]
    , CriticalPointCharacter[#] == 3 && CriticalPointSignature[#] == +1 &
   ];
]
```

```
{0.15949, Null}
```

```
cpsPlus1 // Chop // TableForm
```

```
{}
```

```
AbsoluteTiming[
 cpsPlus3 = Select[
    DeleteDuplicates[
     Select[
      Select[
       ParallelTable[
        Quiet[Check[
          soln = NDSolveValue[
            {
             X'[s] == CriticalPointGradientPlus3[
               {
                g[w, X[s]],
                H[w, X[s]]
               }],
             X[0] == initialValues[[i]]
            },
            {X[s]},
            {s, 0, 10 000}
            , Method → "BDF"
            , MaxSteps → 100
            (*,AccuracyGoal→4,PrecisionGoal→Infinity*)];
          First[soln /. {s → soln[[1, 0, 1, 1, 2]]}]
          ,
          {}
         ]]
        ,
        {i, 1, Length[initialValues], 1}
        , Method → "FinestGrained"
       ]
       , Length[#] == 3 &]
      , ((bbxmin ≤ #[[1]] ≤ bbxmax) &&
          (bbymin ≤ #[[2]] ≤ bbymax) &&
          (bbzmin ≤ #[[3]] ≤ bbzmax)) &]
     , (EuclideanDistance[#1, #2] < 1*^-3) &]
    , CriticalPointCharacter[#] == 3 && CriticalPointSignature[#] == +3 &
   ];
]
```

```
{0.155389, Null}
```

```
cpsPlus3 // Chop // TableForm
```

```
{}
```

```
GraphicsRow[
 {
  ncps // Chop // Sort // TableForm,
  cpsMinus3 // Chop // Sort // TableForm
 }
]
```

0	−1.32401	−0.809514
0	0	0.218693
0	1.32401	−0.809514

{}

```
GraphicsRow[
 {
  bcps // Chop // Sort // TableForm,
  cpsMinus1 // Chop // Sort // TableForm
 }
]
```

0	−1.14644	−0.677648	3	−1
0	1.14644	−0.677648	3	−1

{}

```
GraphicsRow[
 {
  rcps // Chop // Sort // TableForm,
  cpsPlus1 // Chop // Sort // TableForm
 }
]
```

{}

{}

```
GraphicsRow[
 {
  ccps // Chop // Sort // TableForm,
  cpsPlus3 // Chop // Sort // TableForm
 }
]
```

{}

{}

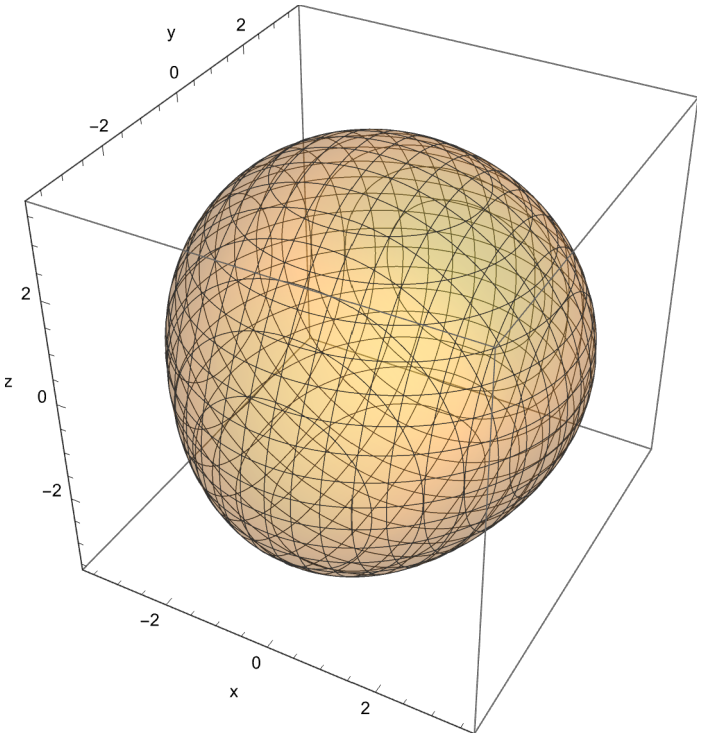
Miscellaneous

Classic examples from traditional QTAIM and Bader's AIM book.

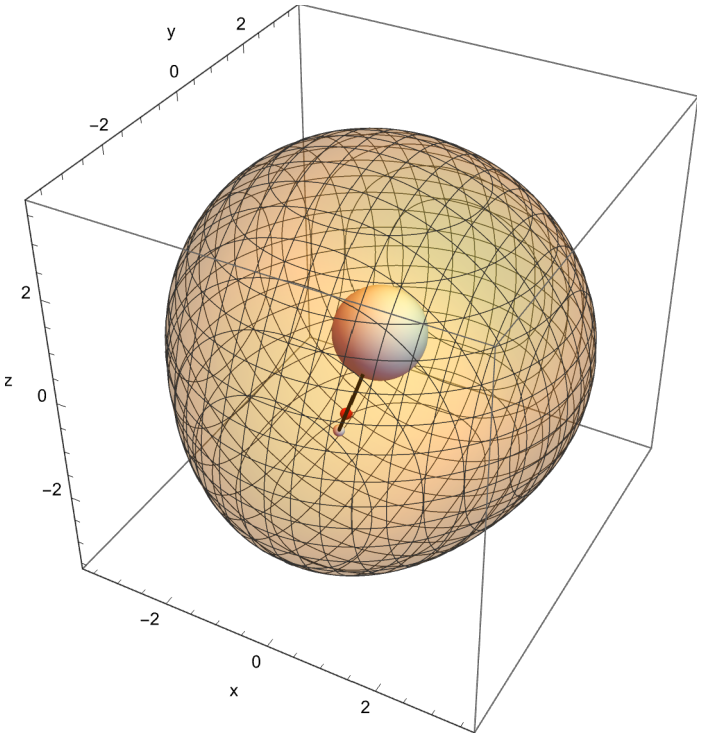
Enclosing envelope at "0.001 a.u."

Mesh:

```
envMesh = ContourPlot3D[
  rho[w, {x, y, z}],
  {x, bbxmin, bbxmax},
  {y, bbymin, bbymax},
  {z, bbzmin, bbzmax},
  AxesLabel → {"x", "y", "z"},
  Contours → {0.001},
  PlotRange → All
  , ContourStyle → Opacity[0.25]
  , ColorFunctionScaling → True
  , BoxRatios → Automatic
 ]
```

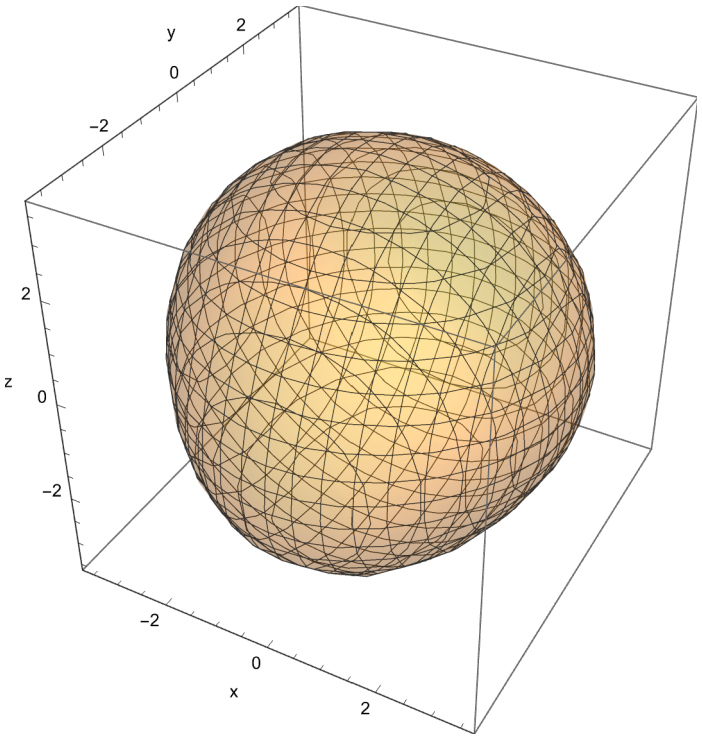


```
Show[
 envMesh,
 graph
]
```

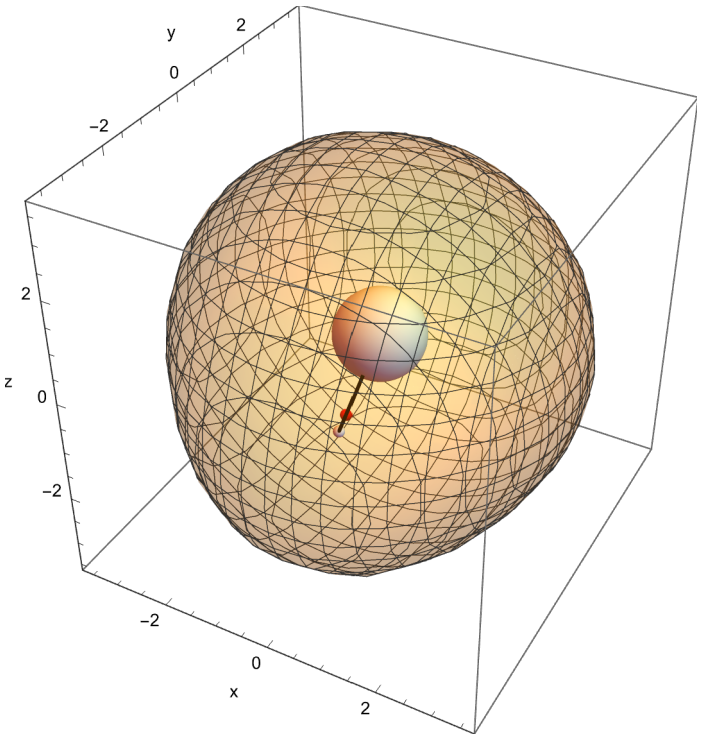


Solid:

```
envRegion = RegionPlot3D[
  rho[w, {x, y, z}] > 0.001,
  {x, bbxmin, bbxmax},
  {y, bbymin, bbymax},
  {z, bbzmin, bbzmax},
  AxesLabel → {"x", "y", "z"},
  PlotRange → All
  , PlotStyle → Opacity[0.25]
  , ColorFunctionScaling → True
  , BoxRatios → Automatic]
```

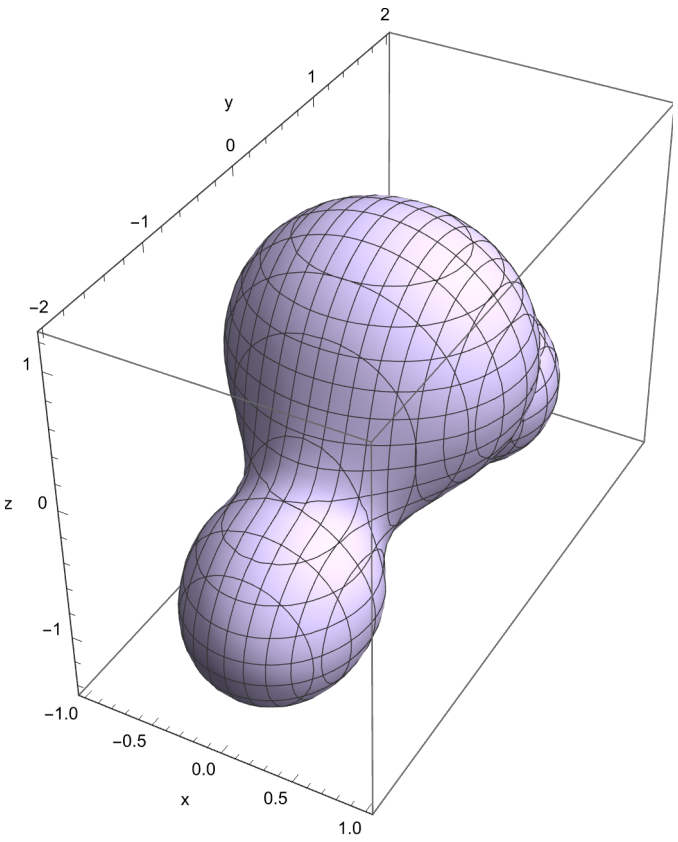


```
Show[
 envRegion,
 graph
]
```



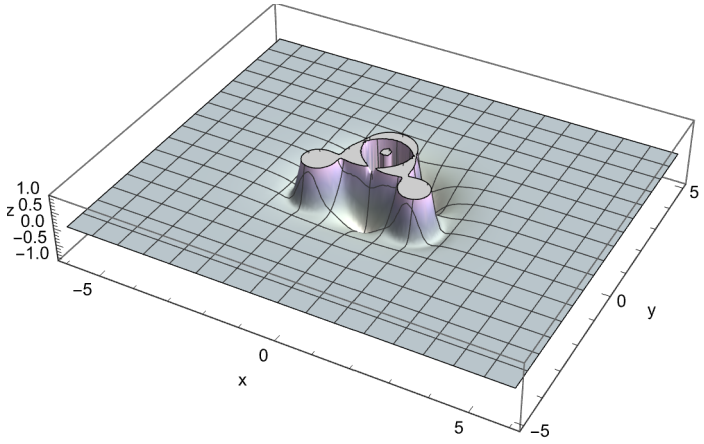
Zero Laplacian Surface:

```
ContourPlot3D[
 -laprho[w, {x, y, z}],
 {x, bbxmin, bbxmax},
 {y, bbymin, bbymax},
 {z, bbzmin, bbzmax},
 AxesLabel → {"x", "y", "z"},
 Contours → {0},
 PlotRange → All
 , ColorFunction → "PearlColors"
 , ColorFunctionScaling → True
 , BoxRatios → Automatic
]
```



Laplacian in a plane, lone pairs are maxima front/right.

```
Plot3D[
 -laprho[w, {0, y, z}],
 {y, bbymin, bbymax},
 {z, bbzmin, bbzmax},
 AxesLabel → {"x", "y", "z"}
 , PlotPoints → 50
 , PlotRange → {Full, Full, {-1*^0, 1*^0}}
 , ColorFunction → "PearlColors"
 , ColorFunctionScaling → True
 , BoxRatios → Automatic
]
```



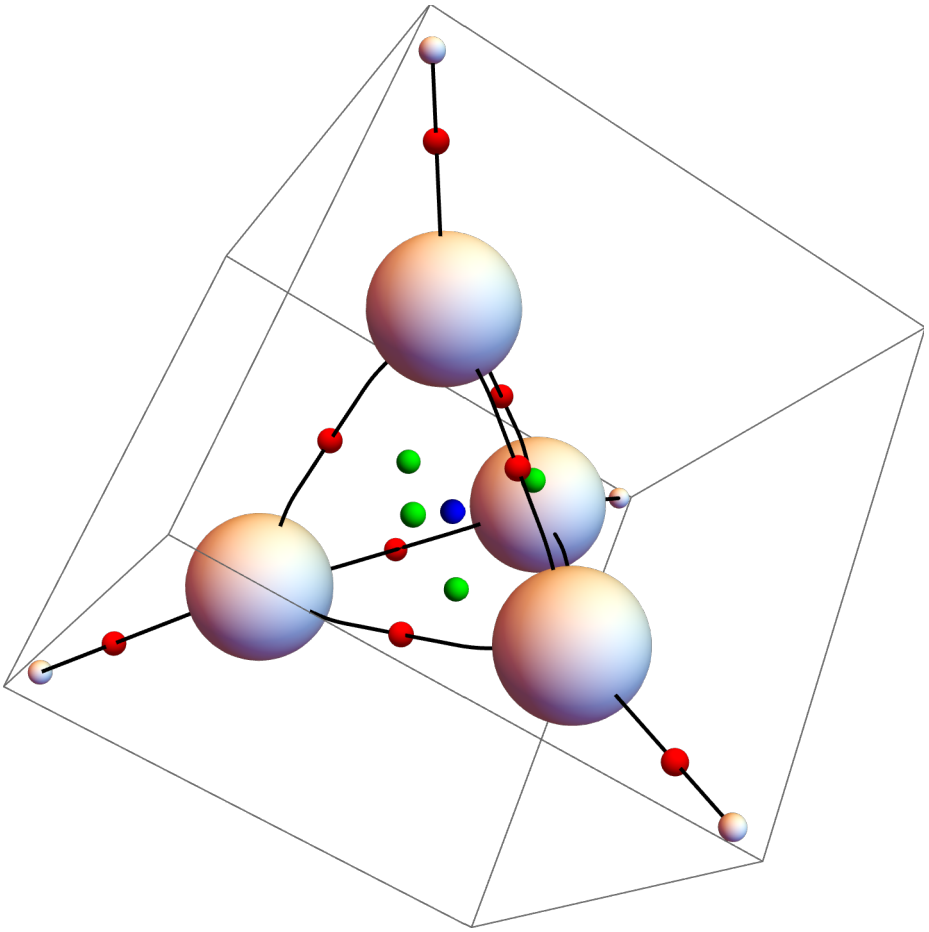
Energy Decomposition over Atomic Basins

```
w["Energy"]
```

```
E = Vnn + Vne + Vee + Tn + Te
```

Tetrahedrane

Showcase tetrahedrane, where all possible critical point types are represented, as a demonstration of fairly comprehensive functionality:



Epilogue

```
Uninstall[link]
ParallelEvaluate[Uninstall[link]]
```