

(continued from previous page)

```

New_Step
  !&{method} &{basis} &{restOfInput}
  %SCF
  EField = &{FFieldStringPlusMinus}
End
&{blocksInput}
Step_End
EPlusMinus.readProperty(propertyName=enPropName);
# -----
# The minus_plus (-+) one
# -----
ReadMOs(1);
New_Step
  !&{method} &{basis} &{restOfInput}
  %SCF
  EField = &{FFieldStringMinusPlus}
End
&{blocksInput}
Step_End
EMinusPlus.readProperty(propertyName=enPropName);
# -----
# And the minus_minus (--) one
# -----
ReadMOs(1);
New_Step
  !&{method} &{basis} &{restOfInput}
  %SCF
  EField = &{FFieldStringMinusMinus}
End
&{blocksInput}
Step_End
EMinusMinus.readProperty(propertyName=enPropName);

  a[i][j] = -(EPlusPlus-EPlusMinus-EMinusPlus+EMinusMinus)/(4*E_Field*E_Field);
EndFor
EndFor

# -----
# Diagonalize
# -----
a.Diagonalize(aEigenValues, aEigenVectors);

# -----
# Do some printing
# -----
print( "\n\n");
print( " ----- \n");
print( "          COMPOUND                      \n");
print( " Numerical calculation of dipole polarizability\n");
print( " ----- \n");
print( " Molecule      : %s\n", molecule);
print( " charge        : %d\n", charge);
print( " Mult          : %d\n", mult);
print( " Method        : %s\n", method);
print( " Basis         : %s\n", basis);
print( " RestOfInput   : %s\n", restOfInput);
print( " BlocksInput   : %s\n", blocksInput);
print( " The electric field perturbation used was:    %.5lf a.u.\n", E_Field);
print( " \n\n");

print( " ----- \n");

```

(continues on next page)

(continued from previous page)

```

print( " Raw electric dipole  polarizability tensor is:\n");
print( " -----\n");
For i from 0 to 2 Do
  print("%13.8lf  %13.8lf  %13.8lf\n", a[i][0], a[i][1], a[i][2]);
EndFor
print( " -----\n");
print("\n");

print( " -----\n");
print( " Raw electric dipole polarizability Eigenvalues\n");
print( " -----\n");
print("%13.8lf  %13.8lf  %13.8lf\n", aEigenValues[0], aEigenValues[1], ↵
↵aEigenValues[2]);
print( " -----\n");
print("\n");

print( " -----\n");
print( " Raw electric dipole polarizability Eigenvectors\n");
print( " -----\n");
For i from 0 to 2 Do
  print("%13.8lf  %13.8lf  %13.8lf\n", aEigenVectors[i][0], aEigenVectors[i][1], ↵
↵aEigenVectors[i][2]);
EndFor

print( "\n a isotropic value : %.5lf\n", ↵
↵(aEigenValues[0]+aEigenValues[1]+aEigenValues[2])/3.0);
print( " -----\n");
print("\n\n");
#
#
# -----
#  Maybe remove unnecesary files
# -----
if (removeFiles) then
  sys_cmd("rm *_Compound* *.bas* ");
EndIf
#
End

```

Comments

In this script we also use the linear algebra *diagonalize* function that is available in *Compound*.

8.4.7 Iterative optimization

Introduction

This is a script that will perform a geometry optimization, then run a frequency calculation and in case there are negative frequencies it will adjust the geometry, based on the Hessian, and optimize again.

Filename

iterativeOptimization.cmp

SCRIPT

```

# Author: Dimitrios G. Liakos and Franke Neese
# Date  : May/June of 2024
#
# ***** DESCRIPTION ↵
↵*****

```

(continues on next page)

(continued from previous page)

```

# iterative Optimization protocol to find structure with no negative
# frequencies (e.g. real minima)
#
# Step 1. Run a single point calculation (we need it for the first property file)
#
# Step 1. Loop and perform calculations with (optimization and frequencies)
#
# Step 2. Check the frequencies. If there are negative ones use the hessian
#         of the appropriate normal mode to adjust the geometry
#
# ----- Variables to adjust (e.g. using 'with') -----
Variable method          = "HF"; # "HF-3c";
Variable MaxNTries       = 25;   # Number of maximum tries
Variable CutOff          = -10.0; # CutOff for a negative frequency
Variable scaling         = 0.6;   # Scaling factor for normal mode
Variable NNegativeTarget = 0;     # Number of negative frequencies we allow
Variable myFilename      = "xyzInput.xyz";
Variable charge          = 0;
Variable multiplicity     = 2;
# -----
# ----- Rest of variables -----
Geometry myGeom;
Variable freqs, modes;
Variable res = -1;
Variable NNegative = 0;
Variable OptDone;

# -----
# Perform a single point calculation. We need it for
# the initial geometry from the property file
# -----
New_Step
  !&{method}
Step_End
myGeom.Read();
myGeom.WriteXYZFile(filename=myFilename);

# -----
# Start a for loop over number of tries
# -----
For itry From 1 To maxNTries Do
  # -----
  # Perform a geometry optimization/Frequency calculation
  # -----
  New_Step
    ! &{method} freq Opt
    *xyzfile &{charge} &{multiplicity} &{myFilename}
  Step_End
  res = freqs.readProperty(propertyName = "THERMO_FREQS");
  res = modes.readProperty(propertyName = "HESSIAN_MODES");
  myGeom.Read();

  # -----
  # check for sufficiently negative frequencies
  # -----
  NNegative = 0;
  For ifreq From 0 to freqs.GetSize()-1 Do
    if ( freqs[ifreq] < CutOff ) then
      myGeom.FollowNormalMode(vibrationSN=ifreq, scalingFactor=scaling);
      NNegative = NNegative + 1;
    endif

```

(continues on next page)

(continued from previous page)

```

endfor
myGeom.WriteXYZFile(filename=myFilename);
If ( NNegative <= NNegativeTarget ) then
    goto OptDone;
endif
endifor

# -----
# Either found correct geometry or reached maximum number of tries.
# -----
OptDone :
if (NNegative > NNegativeTarget) then
    print("ERROR The program did not find a structure with the desired\n number of
    ↳imaginary frequencies.\n There are %9.3lf negative frequencies after %3d steps",
    ↳NNegative, itry);
else
    print("\nSUCCESS optimized structure with (%d) negative\n frequencies found
    ↳after %3d steps", NNegative, itry);
endif
End

```

8.4.8 Gradient extrapolation

Introduction

This script extrapolates the gradient of a molecule. It uses a two point extrapolation where the Hartree-Fock and correlation parts of the gradient are extrapolated separately. This opens the way for geometry optimizations with extrapolated gradients.

Filename

gradientExtrapolation.cmp

SCRIPT

```

# Author: Dimitrios G. Liakos and Frank Neese
# Date : May of 2024
#
# This is a compound file that extrapolates the
# energy gradients to Complete Basis Set Limit (CBS).
#
# STEPS:
# Step1 : Run HF calculation with small basis set
#         Read scfGradX and scfEnX
# Step2 : Run Correlation calculation with small basis set
#         Read totalGradX and totalEnX
# Step3 : Calculate the gradient difference to get
#         the corrGradX (only the correlation part)
# Step4 : Run HF calculation with big basis set
#         Read scfGradY and scfEnY
# Step5 : Run correlation calculation with big basis set
#         Read totalGradY and totalEnY
# Step6 : Calculate the gradient difference with the
#         big basis set to get corrGradY
# Step7 : Evaluate scfGradCBS and scfEnCBS
#         using scfGradX and scfGradY
# Step8 : Evaluate corrGradCBS using
#         corrGradX and corrGradY
# Step9 : Add scfGradCBS and corrGradCBS to get
#         totalGradCBS

```

(continues on next page)

(continued from previous page)

```

# Step10: If needed, create an ORCA engrad file
#
#
# NOTE: It works with an xyz file the name of which we should provide.
#       using the variable initialXYZFilename.
#
# We extrapolate the SCF part using the scheme
#       proposed in: J. Phys. Chem. 129, 184116, 2008
#        $E_{\text{SCF}}(X) = E_{\text{SCF}}(\text{CBS}) + A \exp(-a \sqrt{X})$ 
#
# We extrapolate the correlation part using the scheme
#       proposed in: J. Chem. Phys. 1997, 106, 9639
#        $E_{\text{CBS}}(\text{CORR}) = (X^b * E_X(\text{CORR}) - Y^b * E_Y(\text{CORR})) / (X^b - Y^b)$ 
#
# We use alpha and beta exponents proposed in:
#       J. Chem. Theory Comput., 7, 33-43 (2011)
# ----- Variables -----
# --- Variables to be adjusted (e.g. using 'with' ---
Variable Molecule      = "initial.xyz";    # xyz file of the initial structure
Variable charge         = 0;                # Charge
Variable multiplicity   = 1;                # Spin multiplicity
Variable method         = "MP2";           # The method we use for the
↪ calculation
Variable LowerBasis     = "cc-pVDZ";        # Small basis set
Variable UpperBasis     = "cc-pVTZ";        # Big basis set
Variable restOfInput    = "EnGrad ";        # The rest of the simple input
Variable addCorrelation  = true;            # If we have a correlation part
Variable scfEnPropName  = "MP2_Ref_Energy"; # The name of the property for
↪ the SCF energy
Variable corrEnPropName = "MP2_Corr_Energy"; # The name of the property for
↪ the correlation energy
Variable LowerCardinal  = 2;                # Cardinal number of small basis set
Variable UpperCardinal  = 3;                # Cardinal number of big basis set
Variable alpha          = 4.420;           # Exponent for SCF extrapolation
Variable beta           = 2.460;           # Exponent for correlation
↪ extrapolation
Variable enGradFilename = "result.engrad";   # Filename of the ORCA engrad file
Variable produceEnGradFile = true;          # Produce an ORCA engrad file
# -----
# ----- Rest of the variables -----
Geometry myGeom;
Variable scfGradX, scfGradY;                # SCF Gradients
Variable scfEnX, scfEnY, scfEnCBS;          # SCF energies
Variable corrEnX, corrEnY, corrEnCBS;       # Correlation energies
Variable totalGradX, totalGradY;           # Total Gradients
Variable eX = 0.0;
Variable eY = 0.0;
Variable res = -1;

Variable denominator = 0.0;
Variable gradX = 0.0, gradY = 0.0, gradCBS=0.0;
Variable nAtoms = 0;
Variable EnGradFile;
Variable Cartesians, AtomicNumbers;

# -----
# Step 1. SCF Calculation with small basis set (X)
# -----
New_Step
! HF &{LowerBasis} &{restOfInput}
*xyzfile &{charge} &{multiplicity} &{Molecule}

```

(continues on next page)

(continued from previous page)

```

Step_end
res = scfEnX.readProperty(propertyName="SCF_Energy");
res = scfGradX.readProperty(propertyName="Nuclear_Gradient", Property_Base=true);
myGeom.Read();
nAtoms = myGeom.GetNumOfAtoms();

# -----
# Step 2. Initialize rest of the variables
# -----
Variable corrGradX[3*nAtoms]; # Correlation part of gradient with basis X
Variable corrGradY[3*nAtoms]; # Correlation part of gradient with basis Y
Variable corrGradCBS[3*nAtoms]; # CBS estimation of correlation part of the_
↪gradient
Variable scfGradCBS[3*nAtoms]; # CBS estimation of SCF part of the gradient
Variable totalGradCBS[3*nAtoms]; # CBS estimation of total gradient

# -----
# Step3. Correlation Calculation with small basis set (X)
# -----
if (addCorrelation) then
  New_Step
    ! &{method} &{LowerBasis} &{restOfInput}
  Step_end
    res = scfEnX.readProperty(propertyName=scfEnPropName);
    res = corrEnX.readProperty(propertyName=corrEnPropName);
    res = totalGradX.readProperty(propertyName="Nuclear_Gradient", Property_
↪Base=true);

    # -----
    # Evaluate correlation gradient with small basis set (X)
    # -----
    corrGradX = mat_p_mat(1, totalGradX, -1, scfGradX);
  EndIf

# -----
# Step4. SCF Calculation with large basis set (Y)
# -----
New_Step
  !HF &{UpperBasis} &{restOfInput}
Step_End
res = scfEnY.readProperty(propertyName="SCF_Energy");
res = scfGradY.readProperty(propertyName="Nuclear_Gradient", Property_Base=true);

# -----
# Step5. Correlation calculation with large basis set (Y)
# -----
if (addCorrelation) then
  New_Step
    ! &{method} &{UpperBasis} &{restOfInput}
  Step_end
    res = scfEnY.readProperty(propertyName=scfEnPropName);
    res = corrEnY.readProperty(propertyName=corrEnPropName);
    res = totalGradY.readProperty(propertyName="Nuclear_Gradient", Property_
↪Base=true);

    # -----
    # Evaluate correlation gradient with big basis set Y
    # -----
    corrGradY = mat_p_mat(1, totalGradY, -1, scfGradY);
  EndIf

```

(continues on next page)

(continued from previous page)

```

# -----
# Step6. Extrapolate the SCF part of the gradient and energy
# -----
eX      = exp(-alpha * sqrt(LowerCardinal));
eY      = exp(-alpha * sqrt(UpperCardinal));
denominator = eY-eX;

scfEnCBS = (scfEnX*eY - scfEnY*eX)/(eY-eX);
for i from 0 to scfGradX.GetSize()-1 Do
  gradX = scfGradX[i];
  gradY = scfGradY[i];

  scfGradCBS[i] = (gradX * eY - gradY * eX)/denominator;
endFor

if (addCorrelation) then
  # -----
  # Step7. Extrapolate the correlation part of the gradient and energy
  # -----
  denominator = LowerCardinal^(beta)-(UpperCardinal)^(beta);

  corrEnCBS = (LowerCardinal^(beta)*corrEnX-(UpperCardinal)^(beta)*corrEnY)/
  ↪denominator;
  for i from 0 to scfGradX.GetSize()-1 Do
    gradX = corrGradX[i];
    gradY = corrGradY[i];

    corrGradCBS[i] = (LowerCardinal^(beta)*gradX-(UpperCardinal)^(beta)*gradY)/
    ↪denominator;
  endFor

  # -----
  # Add SCF and correlation part to get total CBS extrapolated values
  # -----
  totalGradCBS = mat_p_mat( 1, scfGradCBS, 1, corrGradCBS);
EndIf

# -----
# Step8. Present the results
# -----
print( "\n\n\n");
print( "-----\n");
print( "          Compound Extrapolation of Gradient          \n");
print( "-----\n");
print( "Number of atoms      : %d\n", nAtoms);
print( "Lower basis set      : %s\n", LowerBasis);
print( "Upper basis set      : %s\n", UpperBasis);
print( "Alpha                : %.2lf\n", alpha);
print( "Beta                 : %.2lf\n", beta);
print( "Lower Cardinal number : %d\n", LowerCardinal);
print( "Upper Cardinal number : %d\n", UpperCardinal);
print( "Method                : %s\n", method);
print( "AddCorrelation        : %s\n", AddCorrelation.GetString());
print( "Produce EnGrad File   : %s\n", produceEnGradFile.GetString());
print( "\n\n");
print( "SCF Energy with small basis set      : %.12e\n", scfEnX);
print( "SCF Energy with big basis set        : %.12e\n", scfEnY);
print( "Extrapolated SCF energy               : %.12e\n", scfEnCBS);
print( "\n\n");
if (addCorrelation) then

```

(continues on next page)

(continued from previous page)

```

print( "Correlation Energy with small basis set : %.12e\n", corrEnX);
print( "Correlation Energy with big basis set   : %.12e\n", corrEnY);
print( "Extrapolated correlation energy        : %.12e\n", corrEnCBS);
print("\n\n");
print( "Total Energy with small basis set : %.12e\n", scfEnX + corrEnX);
print( "Total Energy with big basis set   : %.12e\n", scfEnY + corrEnY);
print( "Extrapolated Total energy        : %.12e\n", scfEnCBS + corrEnCBS);
print("\n\n");
else
print( "Total Energy with small basis set : %.12e\n", scfEnX);
print( "Total Energy with big basis set   : %.12e\n", scfEnY);
print( "Extrapolated Total energy        : %.12e\n", scfEnCBS);
print("\n\n");
EndIf

print( "-----\n");
print( "SCF Gradient with basis set: %s\n", LowerBasis );
print( "-----\n");
print( "Atom      %20s      %20s      %20s\n", "X", "Y", "Z");
for i from 0 to nAtoms-1 Do
  print("%4d      %20lf      %20lf      %20lf\n", i, scfGradX[3*i], scfGradX[3*i+1],
↪scfGradX[3*i+2]);
EndFor
if (addCorrelation) then
  print( "-----\n");
  print( "Correlation Gradient with basis set: %s\n", LowerBasis );
  print( "-----\n");
  print( "Atom      %20s      %20s      %20s\n", "X", "Y", "Z");
  for i from 0 to nAtoms-1 Do
    print("%4d      %20lf      %20lf      %20lf\n", i, corrGradX[3*i],
↪corrGradX[3*i+1], corrGradX[3*i+2]);
  EndFor

  print( "-----\n");
  print( "Total Gradient with basis set: %s\n", LowerBasis );
  print( "-----\n");
  print( "Atom      %20s      %20s      %20s\n", "X", "Y", "Z");
  for i from 0 to nAtoms-1 Do
    print("%4d      %20lf      %20lf      %20lf\n", i, totalGradX[3*i],
↪totalGradX[3*i+1], totalGradX[3*i+2]);
  EndFor
EndIf

print( "-----\n");
print( "SCF Gradient with basis set: %s\n", UpperBasis );
print( "-----\n");
print( "Atom      %20s      %20s      %20s\n", "X", "Y", "Z");
for i from 0 to nAtoms-1 Do
  print("%4d      %20lf      %20lf      %20lf\n", i, scfGradY[3*i], scfGradY[3*i+1],
↪scfGradY[3*i+2]);
EndFor
if (addCorrelation) then
  print( "-----\n");
  print( "Correlation gradient with basis set: %s\n", UpperBasis );
  print( "-----\n");
  print( "Atom      %20s      %20s      %20s\n", "X", "Y", "Z");
  for i from 0 to nAtoms-1 Do
    print("%4d      %20lf      %20lf      %20lf\n", i, corrGradY[3*i],
↪corrGradY[3*i+1], corrGradY[3*i+2]);
  EndFor

```

(continues on next page)

(continued from previous page)

```

print( "-----\n");
print( "Total Gradient with basis set: %s\n", UpperBasis );
print( "-----\n");
print( "Atom      %20s      %20s      %20s\n", "X", "Y", "Z");
for i from 0 to nAtoms-1 Do
  print("%4d      %20lf      %20lf      %20lf\n", i, totalGradY[3*i],
↪totalGradY[3*i+1], totalGradY[3*i+2]);
EndFor
EndIf

print( "-----\n");
print( "Extrapolated SCF part of the Gradient:\n" );
print( "-----\n");
print( "Atom      %20s      %20s      %20s\n", "X", "Y", "Z");
for i from 0 to nAtoms-1 Do
  print("%4d      %20lf      %20lf      %20lf\n", i, scfGradCBS[3*i],
↪scfGradCBS[3*i+1], scfGradCBS[3*i+2]);
EndFor

if (addCorrelation) then
  print( "-----\n");
  print( "Correlation Gradient with basis set:\n" );
  print( "-----\n");
  print( "Atom      %20s      %20s      %20s\n", "X", "Y", "Z");
  for i from 0 to nAtoms-1 Do
    print("%4d      %20lf      %20lf      %20lf\n", i, corrGradCBS[3*i],
↪corrGradCBS[3*i+1], corrGradCBS[3*i+2]);
  EndFor
  print( "-----\n");
  print( "Total Gradient with basis set:\n" );
  print( "-----\n");
  print( "Atom      %20s      %20s      %20s\n", "X", "Y", "Z");
  for i from 0 to nAtoms-1 Do
    print("%4d      %20lf      %20lf      %20lf\n", i, totalGradCBS[3*i],
↪totalGradCBS[3*i+1], totalGradCBS[3*i+2]);
  EndFor
EndIf
print( "-----\n");

if (produceEnGradFile) then
  # -----
  # Read the geometry of the last calculation
  # -----
  myGeom.Read();
  Cartesians = myGeom.GetCartesians();
  atomicNumbers = myGeom.GetAtomicNumbers();
  EnGradFile = openFile(enGradFilename, "w");
  Write2File(EnGradFile, "\n\n\n");
  Write2File(EnGradFile, " %d\n", nAtoms);
  Write2File(EnGradFile, "\n\n\n");
  if (addCorrelation) then
    Write2File(EnGradFile, " %.12lf\n", scfEnCBS + corrEnCBS);
  else
    Write2File(EnGradFile, " %.12lf\n", scfEnCBS);
  EndIf
  Write2File(EnGradFile, "\n\n\n");
  for i from 0 to 3*nAtoms-1 Do
    if (addCorrelation) then
      Write2File(EnGradFile, "      %20.12lf\n", totalGradCBS[i]);
    else

```

(continues on next page)

(continued from previous page)

```

        Write2File(EnGradFile, "          %20.12lf\n", scfGradCBS[i]);
    EndIf
EndFor
Write2File(EnGradFile, "\n\n\n");
for i from 0 to nAtoms-1 Do
    Write2File(EnGradFile, "%5d %12.8lf %12.8lf %12.8lf\n", atomicNumbers[i], ↵
↵cartesians[i][0], cartesians[i][1], cartesians[i][2]);
EndFor
closeFile(EnGradFile);

EndIf

End

```

Comments**8.4.9 BSSE Optimization****Introduction**

This script optimizes the geometry of a molecule using gradients corrected for Basis Set Superposition Error (BSSE) correction. The basic step is the usage of a second script that calculates BSSE corrected gradients.

Filename

BSSEOptimization.cmp

SCRIPT

```

# Author:  Frank Neese and Dimitrios G. Liakos
# Date   :  May of 2024
# -----
#
# This is a script that will use a compound script to
# calculate BSSE corrected gradients and use them
# in combination with ORCA External Optimizer to
# perform a geometry optimization.
#
# We perform the following steps.
# 1. Choose a compound script that calculates the BSSE
#    corrected gradient.
#    We achieve this with the compoundFilename
#
# 2. Create a script to run an ORCA calculation with
#    the external optimizer and the BSSE cprrected
#    gradient. We do that by running a script that runs
#    an ORCA calculation that calculates the gradient
#    and then copy this gradient file back to the expected
#    name
#
# 3. Make a normal ORCA New_Step that calls the external
#    optimizer
#
# NOTE: Depending on the chosen method the property names of
#       myPropName has to be adjusted. For the gradient we do
#       not have this problem because we read the last
#       available in the corresponding property file.
#
# NOTE: Variable baseFilename should have the name of the calling
#       orca input file!
#

```

(continues on next page)

(continued from previous page)

```

# ----- Variables -----
# --- Variables to be adjusted (e.g. using 'with' -----
Variable molecule      = "01.xyz";          # xyz file of the initial structure
Variable method        = "BP86";           # The method we use for the
↳ calculation
Variable basis         = " ";              # The basis set
Variable restOfInput   = "";               # The rest of the simple input
Variable charge        = 0;                # Charge
Variable mult          = 1;                # Spin multiplicity
Variable myPropName    = "SCF_Energy";      # The name of the property for
↳ the energy
variable myFilename    = "compoundBSSE";   # Name for the created xyz files
Variable baseFilename  = "run";
Variable gradCreateFile = "BSSEGradient.cmp"; # The compound script that
↳ extrapolates the gradient
Variable DoOptimization = false;            # Optimize the monomers or not
Variable produceEnGradFile = true;          # Produce an ORCA engrad file
Variable enGradFilename = "result.engrad";  # Filename of the ORCA engrad file
# -----
#
#           Variables for the driver script
Variable createDriverScript = true;          # The shell script driver
Variable driverScript;                     # A script to create the
↳ extrapolated energy gradient
Variable driverScriptName = "runningScript";
Variable submitCommand    = "orca";
# -----
# -----
#
#           Variables for the ORCA input
Variable createORCAInput = true;
Variable orcaInput;          # The ORCA input for the gradient
↳ extrapolation
Variable orcaInputName = "runGradient.inp";
# -----
# -----
# 1. Maybe Create the necessary driver script
#   for the external optimizer and make it executable
#   NOTE: This will depend on the operating system
# -----
if (createDriverScript) then
  driverScript = openFile(driverScriptName, "w");
  write2File(driverScript, "source ~/.bashrc\n");
  write2File(driverScript, "%s %s\n", submitCommand, orcaInputName );
  write2File(driverScript, "cp %s %s_Compound_1_EXT.engrad\n", engradFilename,
↳ baseFilename);
  closeFile(driverScript);
  sys_cmd("chmod +x %s", driverScriptName);
EndIf

# -----
# 2. Maybe Create the ORCA input that will run the
#   compound script for the gradient extrapolation
# -----
if (createORCAInput) then
  orcaInput = openFile(orcaInputName, "w");
  Write2File(orcaInput, "%s_Compound_1_EXT.engrad\n", gradCreateFile);
  Write2File(orcaInput, "  with\n");
  Write2File(orcaInput, "    molecule          = \"%s_Compound_1_EXT.xyz\";\n",

```

(continues on next page)

(continued from previous page)

```

↪baseFilename);
  Write2File(orcaInput, "    charge           = %d;\n",    charge);
  Write2File(orcaInput, "    mutliplicity      = %d;\n",    mult);
  Write2File(orcaInput, "    method        = \"%s\";\n", method);
  Write2File(orcaInput, "    basis          = \"%s\";\n", basis);
  Write2File(orcaInput, "    restOfInput    = \"%s\";\n", restOfInput);
  Write2File(orcaInput, "    myPropName     = \"%s\";\n", myPropName);
  Write2File(orcaInput, "    myFilename     = \"%s\";\n", myFilename);
  Write2File(orcaInput, "    removeFiles    = false;\n");
  Write2File(orcaInput, "    DoOptimization = %s;\n",    DoOptimization.
↪GetString());
  Write2File(orcaInput, "    produceEnGradFile = %s;\n",    produceEnGradFile.
↪GetString());
  Write2File(orcaInput, "    enGradFilename   = \"%s\";\n", enGradFilename);
  Write2File(orcaInput, "End\n");
  closeFile(orcaInput);
EndIf

# -----
# 3. Copy the initial XYZ file to the one needed
#    for the external optimizer
# -----
sys_cmd("cp %s %s_Compound_1_EXT.xyz", molecule, baseFilename);

# -----
# 1. Run the driver ORCA input file that calls the
#    External optimizer
# -----
New_Step
!ExtOpt Opt
*xyzfile &{charge} &{mult} &{baseFilename}_Compound_1_EXT.xyz
%method
  ProgExt "%s\&{driverScriptName}"
End
Step_End

End

```

Comments

The initial structure should contain some ghost atoms.

8.4.10 Umbrella script

Introduction

This script calculates the potential for the “umbrella effect” in NH₃. In addition it locates the minima and maxima in the potential surface.

Filename

Umbrella.cmp

SCRIPT

```

# -----
# Umbrella coordinate mapping for NH3
# Author: Frank Neese
# -----
variable JobName = "NH3-umbrella";

```

(continues on next page)

(continued from previous page)

```

variable amin      = 50.0;
variable amax      = 130.0;
variable nsteps    = 21;
Variable energies[21];

Variable angle;
Variable JobStep;
Variable JobStep_m;
variable step;

Variable method = "BP86";
Variable basis  = "def2-SVP def2/J";

step = 1.0*(amax-amin)/(nsteps-1);

# Loop over the number of steps
# -----
for iang from 0 to nsteps-1 do
  angle      = amin + iang*step;
  JobStep    = iang+1;
  JobStep_m = JobStep-1;
  if (iang>0) then
    Read_Geom(JobStep_m);
    New_step
      ! &{method} &{basis} TightSCF Opt
      %base "&{JobName}.step&{JobStep}"
      %geom constraints
        {A 1 0 2 &{angle} C}
        {A 1 0 3 &{angle} C}
        {A 1 0 4 &{angle} C}
      end
    end

    Step_End
  else
    New_step
      ! &{method} &{basis} TightSCF Opt
      %base "&{JobName}.step&{JobStep}"
      %geom constraints
        {A 1 0 2 &{angle} C}
        {A 1 0 3 &{angle} C}
        {A 1 0 4 &{angle} C}
      end
    end

    * int 0 1
    N 0 0 0 0.0 0.0 0.0
    DA 1 0 0 2.0 0.0 0.0
    H 1 2 0 1.06 &{angle} 0.0
    H 1 2 3 1.06 &{angle} 120.0
    H 1 2 3 1.06 &{angle} 240.0
    *

    Step_End
  endif
  energies[iang].readProperty(propertyName="SCF_ENERGY");
  print(" index: %3d Angle %6.2lf Energy: %16.12lf Eh\n", iang, angle, ↵
↵energies[iang]);
EndFor

# Print a summary at the end of the calculation
# -----

```

(continues on next page)

(continued from previous page)

```

print("////////////////////////////////////////\n");
print("// POTENTIAL ENERGY RESULT\n");
print("////////////////////////////////////////\n");
variable minimum,maximum;
variable Em,E0,Ep;
variable i0,im,ip;
for iang from 0 to nsteps-1 do
  angle  = amin + 1.0*iang*step;
  JobStep = iang+1;
  minimum = 0;
  maximum = 0;
  i0      = iang;
  im      = iang-1;
  ip      = iang+1;
  E0      = energies[i0];
  Em      = E0;
  Ep      = E0;
  if (iang>0 and iang<nsteps-1) then
    Em = energies[im];
    Ep = energies[ip];
  endif
  if (E0<Em and E0<Ep) then minimum=1; endif
  if (E0>Em and E0>Ep) then maximum=1; endif
  if (minimum = 1 ) then
    print(" %3d  %6.21f %16.121f (-)\n",JobStep,angle, E0 );
  endif
  if (maximum = 1 ) then
    print(" %3d  %6.21f %16.121f (+)\n",JobStep,angle, E0 );
  endif
  if (minimum=0 and maximum=0) then
    print(" %3d  %6.21f %16.121f      \n",JobStep,angle, E0 );
  endif
endif
endfor
print("////////////////////////////////////////\n");
End # end of compound block

```

8.4.11 Multi reference

Introduction

This is a script that calculates the atomic electron densities in free atoms and makes a library of them.

Filename

atomDensities.inp

SCRIPT

```

# FN 07/2024
#
# A compound script to run calculation on free atoms
# in order to generate a library of electron densities
#
%compound
  Variable Element = { " ",
                      "H",
                      "Li","Be","B" ,"C" ,"N" ,"O" ,"F" ,"Ne",
                      "He",
                      };
  Variable Nact     = { " ",
                      "1" ,
                      "0",

```

(continues on next page)

(continued from previous page)

```

        "1" , "0" , "3" , "4" , "5" , "6" , "7" , "0"
    };

Variable Norb    = { " ",
                    "1" ,
                    "1" , "0" , "4" , "4" , "4" , "4" , "4" , "0"
                    };

Variable Nroots  = { " ",
                    "1" ,
                    "1" , "0" , "3" , "3" , "1" , "3" , "3" , "0"
                    };

Variable Charge  = { " ",
                    "0" ,
                    "0" , "0" , "0" , "0" , "0" , "0" , "0" , "0"
                    };

Variable Mult    = { " ",
                    "2" ,
                    "2" , "1" , "2" , "3" , "4" , "3" , "2" , "1"
                    };

Variable HFTyp   = { " ",
                    "UHF",
                    "RHF",
                    "CASSCF", "RHF", "CASSCF" , "CASSCF" , "CASSCF" , "CASSCF" ,
                    "CASSCF" , "RHF"
                    };

Variable el;
for el from 1 to Element.GetSize()-1 do
    if (HFTyp[el]="CASSCF") then
        New_Step
        ! cc-pVDZ VeryTightSCF Conv
        %base "atom_{Element[el]}_{Charge[el]}_{Mult[el]}"
        %casscf nel    = &{Nact[el]};
        norb    = &{Norb[el]};
        nroots  = &{Nroots[el]};
        mult    = &{Mult[el]};
        end
        * xyz &{Charge[el]} &{Mult[el]}
        &{Element[el]} 0.0 0.0 0.0
        *
        Step_end
    else
        New_Step
        ! &{HFTyp[el]} cc-pVDZ VeryTightSCF Conv
        %base "atom_{Element[el]}_{Charge[el]}_{Mult[el]}"
        * xyz &{Charge[el]} &{Mult[el]}
        &{Element[el]} 0.0 0.0 0.0
        *
        Step_end
    endif
endfor

endrun;

```

Comments

Here, it's interesting to note that depending on the selected atom, the script either performs a CASSCF calculation, which provides details such as the number of electrons and number of roots, among other parameters, or it carries

out a simple Hartree-Fock calculation.

8.4.12 GoTo

Introduction

This is a brief example demonstrating how the *GoTo* command can be used in *Compound*.

Filename

goTo_Example.inp

SCRIPT

```
# Compound Example on GoTo usage
# Efficient ON/OFF switch

%Compound
  Variable switch="OFF";
  Variable turnOff, turnOn, loopEnd;
  Variable maxIter = 10;
  for i from 0 to maxIter do
    if (switch="ON") then
      GoTo turnOff;
    else
      GoTo turnOn;
    endIf
    turnOff:
      print("Switch: %s\n", switch);
      switch="OFF";
      GoTo loopEnd;
    turnON:
      print("Switch: %s\n", switch);
      switch="ON";
      GoTo loopEnd;
    loopEnd:
  EndFor
End
```


UTILITIES AND VISUALIZATION

9.1 Orbital and Density Plots

There are two types of graphics output possible in ORCA - two dimensional contour plots and three dimensional surface plots. The quantities that can be plotted are the atomic orbitals, molecular orbitals, natural orbitals, the total electron density or the total spin density. The graphics is controlled through the block `%plots`.

9.1.1 Contour Plots

The contour plots are controlled via the following variables

```
%plots
*** the vectors defining the cut plane
v1 0, 0, 0 # pointer to the origin
v2 1, 0, 0 # first direction
v3 0, 1, 0 # second direction
*** alternative to defining vectors. Use atom coordinates
at1 0 # first atom defining v1
at2 2 # second atom defining v2
at3 4 # third atom defining v3
*** resolution of the contour
dim1 45 # resolution in v2-direction
dim2 45 # resolution in v3-direction
*** minimum and maximum values along v2 and v3
min1 -7.0 # min value along v2 in bohr
max1 7.0 # min value along v2 in bohr
min2 -7.0 # min value along v3 in bohr
max2 7.0 # max value along v3 in bohr
***
UseCol true # Use color in the plot (blue=positive,
              # red=negative)
Skeleton true # Draw Skeleton of the molecule of those
              # atoms that are in or close to the cut
              # plane
Atoms true # Draw the atoms that are in the plane as
           # circles
NCont 200 # Number of contour levels.
ICont 0 # Draw NCont equally space contours
      1 # Start with 1/NCont and the double the
        # value for each additional contour
*** the format of the output file
Format Origin # straight ascii files
          HPGL # plotter language files
*** the quantities to plot
MO("MyOrbital-15xy.plt",15,0); # orbital to plot
v3= 0, 0, 1 # change cut plane
MO("MyOrbital-16xz.plt",16,0); # orbital to plot
```

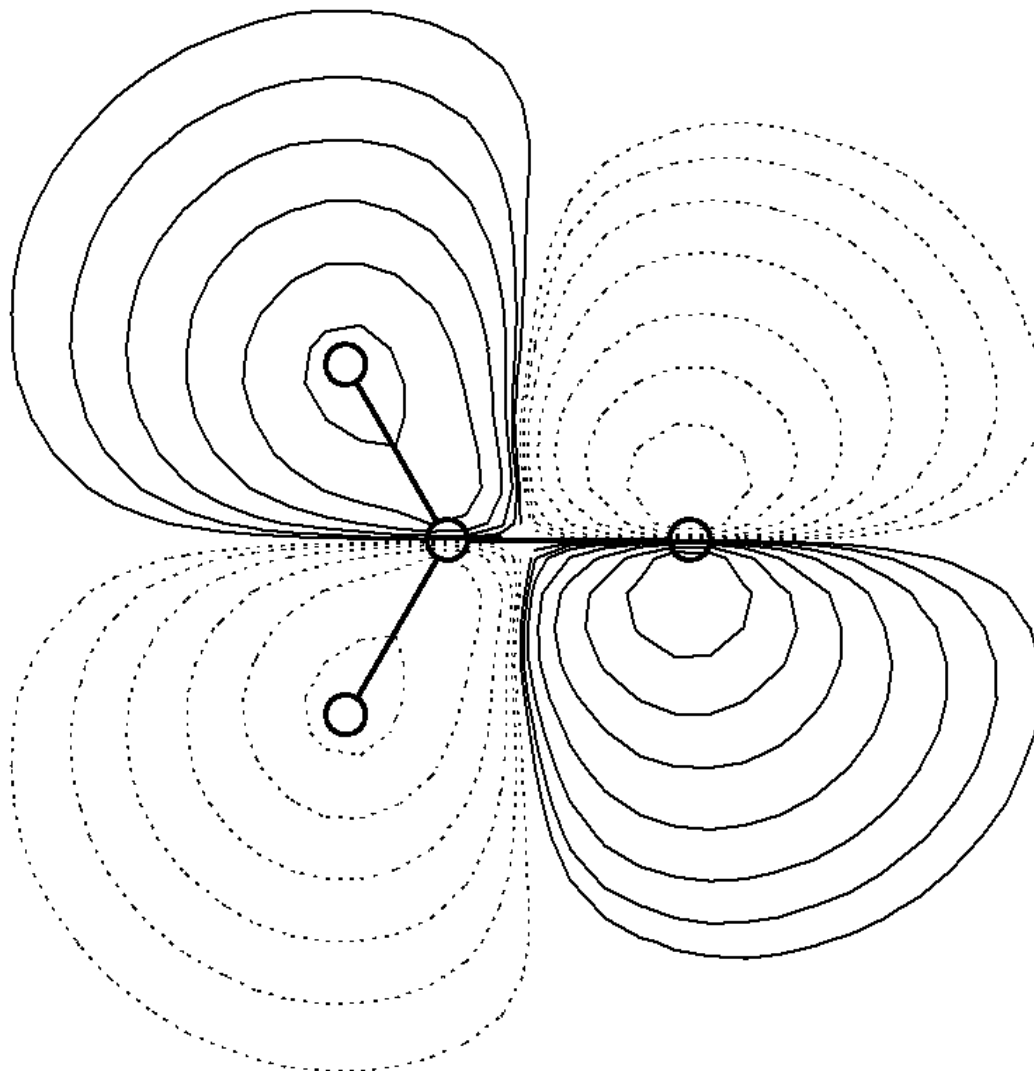
(continues on next page)

(continued from previous page)

```

ElDens("MyElDens.plt");      # Electron density
SpinDens("MySpinDens");      # Spin density
end

```



The input was:

```

v1 = 0, 0, 0; v2 = 1, 0, 0; v3 = 0, 1, 0; min1= -8; max1= 8; min2= -8;
max2= 8; dim1= 50; dim2=50; Format = HPGL; NCont = 200; Icont = 1;
Skeleton= true; Atoms = true; MO("Test-DFT-H2CO+-MO7xy.plt",7,1);

```

NOTE:

- The command `MO("MyOrbital-15xy.plt",15,0);` is to be interpreted as follows: MO means that a MO is to be plotted. "MyOrbital-15xy.plt" is the file to be created. 15 is the number of the MO to be drawn (remember: counting starts at orbital 0!) and 0 is the operator the orbital belongs to. For a RHF (or RKS) calculation there is only one operator which has number 0. For a UHF (or UKS) calculation there are two operators - the spin-up orbitals belong to operator 0 and the spin-down orbitals belong to operator 1. For ROHF calculations there may be many operators but at the end all orbitals will be collected in one set of vectors. Thus the operator is always =0 in ROHF.
- The `ELDENS` (plot of the total electron density) and `SPINDENS` (plot of the total spin density) commands work analogous to the MO with the obvious difference that there is no MO or operator to be defined.

- Analogous to ELDENS and SPINDENS, post-HF densities can be selected using the keyword extended by the respective method. ELDENSMDCI / SPINDENSMDCI will plot the MDCI density, of course only if it is available. ELDENSMP2RE and SPINDENSMP2RE will work with the MP2 relaxed density, while ELDENSMP2UR and SPINDENSMP2UR will yield the MP2 unrelaxed density. The OO-RI-MP2 densities can be requested by ELDENSOO or SPINDENSOO. Similarly, AutoCI relaxed densities can be plotted by using the ELDENSAUTOCIRE and SPINDENSAUTOCIRE keywords, and the unrelaxed densities by using ELDENSAUTOCIUR and SPINDENSAUTOCIUR.
- The UNO option plots natural orbitals of the UHF wavefunction (if they are available). No operator can be given for this command because there is only one set of UHF-NOs. Similarly, using UCO option can be used to plot the UHF corresponding orbitals.
- If the program cannot find the plot module (“Bad command or filename”) try to use `ProgPlot="orca_plot.exe"` in the `%method` block or point to the explicit path.
- The defining vectors `v2` and `v3` are required to be orthonormal. The program will use a Schmidt orthonormalization of `v3` with respect to `v2` to ensure orthonormality. If you do not like this make sure that the input vectors are already orthogonal.
- `at1`, `at2` and `at3` can be used instead of `v1`, `v2` and `v3`. In this case say `v1` is taken as the coordinates of atom `at1`. Mixed definitions where say `v2` is explicitly given and say `v3` is defined through `at3` are possible. A value of -1 for `at1`, `at2` and `at3` signals that `at1`, `at2` and `at3` are not to be used. This type of definition may sometimes be more convenient.
- Variables can be assigned several times. The “actual” value a variable has is stored together with the command to generate a plot (MO, ELDENS or SPINDENS). Thus after each plot command the format or orientation of the plot can be changed for the next one.
- The `Origin` format produces a straightforward ASCII file with x, y and z values that can be read into your favorite contour plot program or you could write a small program that reads such files and converts them to whatever format is more appropriate for you.
- I usually use Word for Windows to open the HPGL files which appears to work fine. Double clicking on the graphics will allow modification of linewidth etc. For some reason that is not clear to me some graphics programs do not like the HPGL code that is produced by ORCA. If you are an HPGL expert and you have a suggestion - let me know.

9.1.2 Surface Plots

General Points

Surface plots can, for example, be created through an interface to Leif Laaksonen’s *gOpenMol* program. This program can be obtained free of charge over the internet. It runs on a wide variety of platforms, is easy to use, produces high quality graphics and is easy to interface¹ - thank you Leif for making this program available!

The relevant [PLOTS] section looks like this:

```
%output
  XYZFile true
end

%plots
dim1 45 # resolution in x-direction
dim2 45 # resolution in y-direction
dim3 45 # resolution in z-direction
min1 -7.0 # x-min value in bohr
max1 7.0 # x-min value in bohr
min2 -7.0 # y-min value in bohr
max2 7.0 # y-max value in bohr
```

(continues on next page)

¹ There were some reports of problems with the program on Windows platforms. Apparently it is better to choose the display settings as “true color 32 bit” rather than “high 16 bit”. Thanks to Thomas Brunold!

(continued from previous page)

```

min3 -7.0 # z-min value in bohr
max3  7.0 # z-max value in bohr
Format  gOpenMol_bin      # binary *.plt file
        gOpenMol_ascii   # ascii *.plt file
        Gaussian_Cube     # Gaussian-cube format
                        # (an ASCII file)
MO("MyOrbital-15.plt",15,0); # orbital to plot
MO("MyOrbital-16.plt",16,0); # orbital to plot
UNO("MyUNO-48.plt",48);    # UHF-NO to plot
ElDens("MyElDens.plt");    # Electron density
SpinDens("MySpinDens.plt"); # Spin density
end

```

Note

- it is admittedly inconvenient to manually input the dimension of the cube that is used for plotting. If you do nothing such that `min1 = max1 = min2 = max2 = min3 = max3=0` then the program will try to be smart and figure out a good cube size by itself. It will look at the minimum and maximum values of the coordinates and then add 7 bohrs to each dimension in the hope to properly catch all wavefunction tails.









Sometimes you will want to produce orbital plots after you looked at the output file and decided which orbitals you are interested in. In this case you can also run the `orca_plot` program in a crude interactive form by invoking it as:

```
orca_plot MyGBWFile.gbw -i
```

This will provide you with a subset of the capabilities of this program but may already be enough to produce the plots you want to look at. Note that for the name of the GBW-file you may as well input files that result from natural orbitals (normally `*.uno`), corresponding orbitals (normally `*.uco`) or localized orbitals (normally `*.loc`). Once in the interactive program, by entering '1' for 'Enter type of plot,' you will access a list of available plot capabilities relevant to your current calculation file (`MyGBWFile.gbw`):

```
-----
Plot-Type is presently: 1
-----
```

```
Searching for Ground State Electron or Spin Densities:      ...
-----
```

1 -	molecular orbitals		
2 -	(scf) electron density	(scfp) 
→) => AVAILABLE		
3 -	(scf) spin density	(scfr) 
→) => AVAILABLE		
4 -	natural orbitals		
5 -	corresponding orbitals		
6 -	atomic orbitals		
7 -	mdci electron density	(mdcip) 
→) - NOT AVAILABLE		
8 -	mdci spin density	(mdcir) 
→) - NOT AVAILABLE		
9 -	OO-RI-MP2 density	(pmp2re) 
→) - NOT AVAILABLE		
10 -	OO-RI-MP2 spin density	(pmp2ur) 
→) - NOT AVAILABLE		
11 -	MP2 relaxed density	(pmp2re) 
→) - NOT AVAILABLE		
12 -	MP2 unrelaxed density	(pmp2ur) 
→) - NOT AVAILABLE		

(continues on next page)