Alexandre Sureda Croguennoc - Restricted Hartree-Fock of HeH+
Mathematica code for HeH+ using basis functions from Szabo book, more efficient than the Python code because in the *Mathematica* environment is more easy to do do expensive calculations with less computation time because the way of programming is different than Fortran, Python, etc. The most beatiful think in this code is the definition of bilinear function, and quadrilinear function with the restrictions of this operator (extracted from the Introduction of *Mathematica* book) to calculate the matrix integrals. Also the computation time is less because in *Mathematica* we can do the same calculations like in Python but with less loops making the time of the calculation more efficient.

:

```
In[1]:= ClearAll[GaussianPrimitive, MakeBilinearFunction,
          OverlapMatrix, KineticMatrix, Func, NuclearConverge, NuclearDiverge,
          CreateNuclearMatrix, MakeQuadrilinearFunction, TwoElectronConverge,
          TwoElectronDiverge, TwoElectronMatrix, CreateGMatrix];
        (*make name into a functino that's linear in its first 2 inputs*)
        MakeBilinearFunction[name_] := Module[{},
          ClearAll[name];
          name[c_?NumberQ * f_GaussianPrimitive, h_, theRest___] :=
           c * name[f, h, theRest];
          name[h_, c_?NumberQ * f_GaussianPrimitive, theRest___] :=
           c * name[h, f, theRest];
          name[f_ + g_, h_, theRest___] := name[f, h, theRest] + name[g, h, theRest];
          name[h_, f_ + g_, theRest___] := name[h, f, theRest] + name[h, g, theRest];
         ]
        MakeBilinearFunction[OverlapMatrix];
        OverlapMatrix[GaussianPrimitive[α_, RA_], GaussianPrimitive[β_, RB_]] :=
          (2 α/π)^(3/4) (2 β/π)^(3/4) (π/(α + β))^(3/2) Exp[-(α β)/(α + β) Norm[RA - RB]^2];
        MakeBilinearFunction[KineticMatrix];
        KineticMatrix[GaussianPrimitive[α_, RA_], GaussianPrimitive[β_, RB_]] :=
          (2 α/π)^(3/4) (2 β/π)^(3/4) ((α β)/(α + β)) (3 - (2 α β)/(α + β) Norm[RA - RB]^2)
           (π/(α + β))^(3/2) Exp[-(α β)/(α + β) Norm[RA - RB]^2];
        Func[t_] := 1/2 (π/t)^(1/2) Erf[Sqrt[t]];
        MakeBilinearFunction[NuclearConverge];
        NuclearConverge[GaussianPrimitive[α_, RA_],
           GaussianPrimitive[β_, RB_], ZC_, RC_] := -(2 α/π)^(3/4) (2 β/π)^(3/4) ((2 π)/(α + β) (ZC))
           Exp[-(α β)/(α + β) Norm[RA - RB]^2] Func[(α + β) Norm[((α RA + β RB)/(α + β)) - (RC)]^2];
        MakeBilinearFunction[NuclearDiverge];
        NuclearDiverge[GaussianPrimitive[α_, RA_], GaussianPrimitive[β_, RB_],
           ZC_, RC_] := -2 (2 α/π)^(3/4) (2 β/π)^(3/4) Exp[-(α β)/(α + β) Norm[RA - RB]^2] (π/(α + β) ZC);
        (*return a list of length numberOfNuclei,
        consisting of nuclear attraction matrices*)
```

```
CreateNuclearMatrix[] :=
  Module[{nucleusCounter, currentCharge, currentCoords, ret},
   noOfNuclei = numberOfNuclei;
   charge = nuclearCharge;
   coords = nuclearCoordinates;
   ret = Table[0, {noOfNuclei}];
   For[nucleusCounter = 1, nucleusCounter ≤ noOfNuclei, nucleusCounter++,
    currentCharge = charge[[nucleusCounter]];
    currentCoords = coords[[nucleusCounter]];
    ret[[nucleusCounter]] = Table[If[(i == j) && (i == nucleusCounter),
       NuclearDiverge[ϕ[[i]], ϕ[[j]], currentCharge, currentCoords],
       NuclearConverge[ϕ[[i]], ϕ[[j]], currentCharge, currentCoords]
      ],
      {i, 1, noOfNuclei},
      {j, 1, noOfNuclei}
     ];
   ];
   Return[ret];
  ];
(*make name into a function that's linear in its 4 inputs*)
MakeQuadrilinearFunction[name_] := Module[{},
   ClearAll[name];
   name[c_?NumberQ * f_GaussianPrimitive, theRest___] := c * name[f, theRest];
   name[f_ + g_, theRest___] := name[f, theRest] + name[g, theRest];
   name[h_, c_?NumberQ * f_GaussianPrimitive, theRest___] :=
    c * name[h, f, theRest];
   name[h_, f_ + g_, theRest___] := name[h, f, theRest] + name[h, g, theRest];
   name[m_, h_, c_?NumberQ * f_GaussianPrimitive, theRest___] :=
    c * name[m, h, f, theRest];
   name[m_, h_, f_ + g_, theRest___] :=
    name[m, h, f, theRest] + name[m, h, g, theRest];
   name[n_, m_, h_, c_?NumberQ * f_GaussianPrimitive, theRest___] :=
    c * name[n, m, h, f, theRest];
   name[n_, m_, h_, f_ + g_, theRest___] :=
    name[n, m, h, f, theRest] + name[n, m, h, g, theRest];
  ];
MakeQuadrilinearFunction[TwoElectronConverge];
TwoElectronConverge[GaussianPrimitive[α_, RA_], GaussianPrimitive[β_, RB_],
   GaussianPrimitive[γ_, RC_], GaussianPrimitive[δ_, RD_]] :=
```

$$\left(\frac{2\,\alpha}{\pi}\right)^{3/4}\left(\frac{2\,\beta}{\pi}\right)^{3/4}\left(\frac{2\,\gamma}{\pi}\right)^{3/4}\left(\frac{2\,\delta}{\pi}\right)^{3/4}\frac{2\,\pi^{5/2}}{(\alpha+\beta)\;(\gamma+\delta)\;\sqrt{\alpha+\beta+\gamma+\delta}}$$

$$\text{Exp}\left[-\frac{\alpha\,\beta}{\alpha+\beta}\,\text{Norm}[\text{RA}-\text{RB}]^{2}-\frac{\gamma\,\delta}{\gamma+\delta}\,\text{Norm}[\text{RC}-\text{RD}]^{2}\right]$$

$$\text{Func}\left[\frac{(\alpha+\beta)\;(\gamma+\delta)}{\alpha+\beta+\gamma+\delta}\,\text{Norm}\left[\left(\frac{\alpha\,\text{RA}+\beta\,\text{RB}}{\alpha+\beta}\right)-\left(\frac{\gamma\,\text{RC}+\delta\,\text{RD}}{\gamma+\delta}\right)\right]^{2}\right];$$

```
MakeQuadrilinearFunction[TwoElectronDiverge];
TwoElectronDiverge[GaussianPrimitive[α_, RA_], GaussianPrimitive[β_, RB_],
   GaussianPrimitive[γ_, RC_], GaussianPrimitive[δ_, RD_]] :=
```

$$\left(\frac{2\,\alpha}{\pi}\right)^{3/4}\left(\frac{2\,\beta}{\pi}\right)^{3/4}\left(\frac{2\,\gamma}{\pi}\right)^{3/4}\left(\frac{2\,\delta}{\pi}\right)^{3/4}\frac{2\,\pi^{5/2}}{(\alpha+\beta)\;(\gamma+\delta)\;\sqrt{\alpha+\beta+\gamma+\delta}}$$

```
    Exp[- αβ/(α + β) Norm[RA - RB]^2 - γδ/(γ + δ) Norm[RC - RD]^2];
CreateGMatrix[] := Return[Table[Sum[PMatrix[[λ, σ]]
       (TwoElectronMatrix[{μ, ν}, {σ, λ}] - 1 / 2 TwoElectronMatrix[{μ, λ}, {σ, ν}]),
      {λ, sizeOfBasis}, {σ, sizeOfBasis}]
     , {μ, sizeOfBasis}, {ν, sizeOfBasis}]];

ClearAll[numberOfNuclei, numberOfElectrons, sizeOfBasis, ϕ, nuclearCharge,
  nuclearCoordinates, SMatrix, HCoreMatrix, XMatrix, PMatrix, GMatrix, newEnergy,
  oldEnergy, loopCounter, FPrimeMatrix, CPrimeMatrix, εMatrix, CMatrix, FMatrix];
```

Actual calculations:

```
ClearAll[numberOfNuclei, numberOfElectrons, sizeOfBasis,
  ϕ, nuclearCharge, nuclearCoordinates, SMatrix, HCoreMatrix,
  XMatrix, PMatrix, GMatrix, newEnergy, oldEnergy, loopCounter,
  FPrimeMatrix, CPrimeMatrix, εMatrix, CMatrix, FMatrix];
(*TwoElectronMatrix needs to be cleared every iteration because we directly
 associate numerical values with the name (rather than through a function),
and these values are different for different parameters *)
ClearAll[TwoElectronMatrix];
SetAttributes[TwoElectronMatrix, Orderless];
TwoElectronMatrix[{μ_, ν_}, {λ_, σ_}] /; ν < μ :=
  TwoElectronMatrix[{ν, μ}, {λ, σ}];
TwoElectronMatrix[{μ_, ν_}, {λ_, σ_}] := Module[{ret},
   If[(μ == λ) && (ν == σ),
    ret = TwoElectronDiverge[ϕ[[μ]], ϕ[[ν]], ϕ[[λ]], ϕ[[σ]]];
    ,
    ret = TwoElectronConverge[ϕ[[μ]], ϕ[[ν]], ϕ[[λ]], ϕ[[σ]]];
   ];
   TwoElectronMatrix[{μ, ν}, {λ, σ}] = ret;
   Return[ret];
  ];
(*----------set up for HeH+: 1st nucleus is He,2nd nucleus is H-----------*)
numberOfNuclei = 2;
numberOfElectrons = 2;
sizeOfBasis = 2; (*=K*)
ϕ = Table[0, {sizeOfBasis}];
nuclearCharge = {2., 1.};
nuclearCoordinates = {{0, 0, 0}, {1.4632, 0, 0}};
(*-----basis needs to be named ϕ*)
ϕ[[2]] = 0.444635 GaussianPrimitive[0.168856, nuclearCoordinates[[2]]] +
   0.535328 GaussianPrimitive[0.623913, nuclearCoordinates[[2]]] +
   0.154329 GaussianPrimitive[3.42525, nuclearCoordinates[[2]]];
ϕ[[1]] = 0.444635 GaussianPrimitive[0.48084429026249986,
     nuclearCoordinates[[1]]] +
   0.535328 GaussianPrimitive[1.7766911481187495, nuclearCoordinates[[1]]] +
   0.154329 GaussianPrimitive[9.753934615874998, nuclearCoordinates[[1]]];
(*-----Step 3: Calculate S,H^core,and two-electron integrals:*)
Module[{VMatrix, TMatrix},
  SMatrix = Outer[OverlapMatrix, ϕ, ϕ];
  TMatrix = Outer[KineticMatrix, ϕ, ϕ];
  (*VMatrix=
    CreateNuclearMatrix[numberOfNuclei,nuclearCharge,nuclearCoordinates];*)
   VMatrix = CreateNuclearMatrix[];
   HCoreMatrix = Plus @@ VMatrix + TMatrix;
   (*calculate all two-electron integrals*)
   Tuples[Range[sizeOfBasis], 4]
     (*form all possible combination of indices of basis functions*)
    /. {x_, y_, z_, t_} :→ {TwoElectronMatrix[{x, y}, {z, t}], {x, y, z, t}}
     (*force evaluation of two-
    electron integrals for all those  combinations*);
  ];
 (*-----Step 4:
  Diagonalize overlap matrix S and obtain transformation matrix X*)
 Module[{UMatrix, eig, sInverseSqrtMatrix},
   UMatrix = Transpose[Eigenvectors[SMatrix]];
   eig = Eigenvalues[SMatrix];
   (*form matrix s^-1/2 :*)
   sInverseSqrtMatrix = DiagonalMatrix[1 / Sqrt[eig]];
   XMatrix = UMatrix.sInverseSqrtMatrix;
  ];
 (*-----Step 5: Guess the density matrix P:*)
 PMatrix = Table[0, {sizeOfBasis}, {sizeOfBasis}];
 (*-----Step 6:
  Calculate matrix G from density matrix P and two-electron integrals:*)
 GMatrix = CreateGMatrix[];
 (*----Step 7: Obtain Fock matrix F and calculate the energy E0:*)
 FMatrix = HCoreMatrix + GMatrix;
 newEnergy = 1/2 Tr[PMatrix.(HCoreMatrix + FMatrix)];
 Print["Initial energy: ", newEnergy];
 oldEnergy = -10 000;
 (*no electronic energy can be this large. Just to
  make sure that the while loop executes at least once*)
 loopCounter = 0;
 While[Abs[oldEnergy - newEnergy] > 10^-6, (*testing convergence*)
   oldEnergy = newEnergy;
   loopCounter++;
   (*----Step 8: Calculate transformed matrix F'*)
   FPrimeMatrix = ConjugateTranspose[XMatrix].FMatrix.XMatrix;
   (*----Step 9: Diagonalize F' to obtain C' and ε*)
   CPrimeMatrix = Transpose[Eigenvectors[FPrimeMatrix]];
   εMatrix = DiagonalMatrix[Eigenvalues[FPrimeMatrix]];
   (*----Step 10: Calculate C matrix*)
   CMatrix = XMatrix.CPrimeMatrix;
   (*----Step 11: Calculate new P matrix*)
   PMatrix =
    Table[2 Plus @@ (Take[CMatrix[[i]], numberOfElectrons / 2] Take[CMatrix[[j]],
          numberOfElectrons / 2]), {i, sizeOfBasis}, {j, sizeOfBasis}];
   (*-----Step 12: Calculate GTest:*)
   GMatrix = CreateGMatrix[];
   (*----Step 13: Obtain Fock matrix FTest and calculate the energy E0:*)
   FMatrix = HCoreMatrix + GMatrix;
   newEnergy = 1/2 Tr[PMatrix.(HCoreMatrix + FMatrix)];
  Print["Iteration ", loopCounter, " Energy: ", newEnergy];
 ];
Print["--------------------------------"];
electronicEnergy = newEnergy;
Print["Electronic Energy: ", N[electronicEnergy, 10]];
(*calculate the  total nuclear repulsion:*)
nuclearRepulsionEnergy = Plus @@ (Subsets[Range[numberOfNuclei], {2}]
     (*form a list containing all unique pairs of nuclei*)
     /. {x_Integer ? Positive, y_Integer ? Positive} :>
      (nuclearCharge[[x]] nuclearCharge[[y]]) /
       Norm[nuclearCoordinates[[x]] - nuclearCoordinates[[y]]]);
(*then replace each element of the list by the corresponding
 nuclear repulsion*)
Print["Nuclear repulsion: ", nuclearRepulsionEnergy];
totalEnergy = electronicEnergy + nuclearRepulsionEnergy;
Print["Total molecular energy: ", totalEnergy];
Print["Matrix C: ", MatrixForm[CMatrix]];
Print["Matrix ε: ", MatrixForm[εMatrix]];
```

```
Initial energy: 0.

Iteration 1 Energy: -4.14125

Iteration 2 Energy: -4.22409

Iteration 3 Energy: -4.22489

Iteration 4 Energy: -4.22489

Iteration 5 Energy: -4.22489

--------------------------------

Electronic Energy: -4.22489

Nuclear repulsion: 1.36687

Total molecular energy: -2.85802
```

Matrix C: $\begin{pmatrix} 0.803525 & -0.780612 \\ 0.334601 & 1.06914 \end{pmatrix}$

Matrix ε: $\begin{pmatrix} -1.5938 & 0. \\ 0. & -0.081114 \end{pmatrix}$