Exercise Ex4

November 3, 2025

Student: Romain Couyoumtzélis Sciper: 340933

Please use this template to submit your answers.
If you had to modify code from the notebook, please include the modified code in your submission either as screenshot or in a

```
\begin{lstlisting}[language=Python]
\end{lstlisting}
```

environment.
You only need to include the code cells that you modified.
Note, that references to other parts of the documents aren't resolved in this template and will show as `??`. Check the text of the exercises on website for the reference

Exercise 1
Define S_{ij} using Dirac notation.

$$S_{ij} = \langle i|j\rangle = \langle \phi_i|\phi_j\rangle$$

Exercise 2
For an orthonormal basis, what does the overlap integral array, `S`, look like?

For an orthonormal basis, $S_{ij} = \delta_{ij}$. `S` is the identity matrix. It has the dimension equals to the size of our basis set.

Exercise 3
Use `B` and `B_dagger` and the matrix rules above to calculate the matrix `S`.

```
S = B_dagger.dot(B)
```

Exercise 4
Describe how the notation of the `np.einsum` command correlates to the implicit summation formula written above.

```
S=numpy.einsum('ip,iq->pq',B.conj(),B)
```

It means that we indicate to numpy a repeated index i on which we will make the implicit summation. Indeed we will sum on i tensor B^*_{ip} with tensor B_{iq} and reorganize the result as a new tensor S_{pq}. This corresponds to $\sum_i B^*_{ip} B_{iq} = S_{pq}$ which is exactly the overlap matrix.

Exercise 5
Use the function `np.einsum()` to calculate the matrix `S`, and confirm that your answer is the same as above.

```
S = np.einsum('ip,iq->pq',B.conj(),B) #<your formula here>
print(F'S from Einstein notation:\n{S}')
```

```
S from Einstein notation:  [[1.  0.]  [0.  1.]]
```

Exercise 6
Propose a different orthonormal basis, modify `phi1` and `phi2`, and verify that `S` still has the same form. There are infinitely many choices. It isn't complex... or *is* it?!

```
# two orthonormal vectors
phi1 = np.array([np.cos(.5), np.sin(.5)]) #insert here
phi2 = np.array([np.sin(.5), -np.cos(.5)]) #insert here

# construct the overlap matrix from matrix of basis vectors
vector_length = phi1.size #length of the vector space
phi1_column = phi1.reshape(vector_length,1) #this makes phi a
    column vector
phi2_column = phi2.reshape(vector_length,1)

# put together (concatenate) the vectors into the matrix B
B = np.concatenate((phi1_column,phi2_column),axis=1)
print(F'The matrix B:\n{B}')

B_dagger = B.conj().T
print(F'The matrix B^\\dagger:\n{B_dagger}')

# calculate S from matrix of basis vectors
S = np.einsum('ip,iq->pq',B.conj(),B)
print(F'S from Einstein notation:\n{S}')
```

```
The matrix B: [[ 0.87758256 0.47942554] [ 0.47942554 -0.87758256]]
The matrix B^†:  [[ 0.87758256 0.47942554] [ 0.47942554 -0.87758256]]
S from Einstein notation:  [[1.  0.]  [0.  1.]]
```

Exercise 7
How many electrons are there in total in H_2O? How many occupied molecular orbitals would you expect?

There are 10 electrons in H_2O molecule. They are occupied orbitals 5 orbitals. We have one first orbital for the core electrons, which are paired, and are from O. Then O has two unpaired electrons which will form each a covalent bond with each electron from H atoms, leading to 1 orbital each. Finally, O has two lone pairs for which we allocate to each an orbital.

Exercise 8
Explain the shape (number of rows and columns) of `S` in terms of the AO basis set we chose.

```
# Construct a molecular integrals object
mints = psi4.core.MintsHelper(wfn.basisset())

# Overlap matrix as a psi4 Matrix object
S_matrix = mints.ao_overlap()

# Overlap matrix converted into an ndarray
S = np.asarray(S_matrix)

print(F'Shape of S is {S.shape}')
```

```
Shape of S is (7, 7)
```

Since we have 7 basis fonctions, we need to calculate the overlap S_{ij} for all pairs $\{i, j\} \in \{1, ..., 7\}^2$. Leading to a matrix of dimension 7×7. STO-3G gives 5 basis functions for O ($1s^2 2s^2 2p^4$, one basis functions for each orbital) and 1 per H atoms ($1s^2$). Therefore :

$$S = \begin{pmatrix} \langle\phi_1|\phi_1\rangle & \langle\phi_1|\phi_2\rangle & \langle\phi_1|\phi_3\rangle & \cdots \\ \langle\phi_2|\phi_1\rangle & \langle\phi_2|\phi_2\rangle & \langle\phi_2|\phi_3\rangle & \cdots \\ \langle\phi_3|\phi_1\rangle & \langle\phi_3|\phi_2\rangle & \langle\phi_3|\phi_3\rangle & \cdots \\ \vdots & \vdots & \vdots & \ddots \\ & & & \langle\phi_7|\phi_7\rangle \end{pmatrix}$$

Exercise 9
Based on your observations of `S` in the AO basis, answer the following questions

1. What do the diagonal elements of `S` indicate?
2. What do the off-diagonal elements of `S` indicate?

3. Does the Gaussian atomic orbital basis set form an orthonormal basis?

1. The diagonal terms are equal to 1, the basis sets is normalized.
2. Most of the terms are 0 but some terms are of order 10^{-1} and 10^{-2} indicating that the overlap between two different atomic orbitals is not always 0 and that not all states are orthogonal.
3. The Gaussian atomic orbital basis set does not form a complete orthonormal basis set due to significant overlap between some orbitals. `S` will have to be considered for further calculations.

Exercise 10
Does the result of your extra evaluation agree with what you determined previously?

```
isBasisOrthonormal(S)
```

```
  Q:(T/F) The AO basis is orthonormal?  A: False
False
```

This confirms the fact that the gaussian atomic orbital basis set does not form a complete ortho-normal basis set.

Exercise 11
Use the function `np.linalg.inv()` to calculate the inverse of `S`, and the function `splinalg.sqrtm()` to take its (matrix) square root. Execute the code below and examine the matrix `A`.

```
S_inverse = np.linalg.inv(S)

A = splinalg.sqrtm(S_inverse)

peak(A)
```

```
  Here is a peak at the first 4 x 4 elements of the matrix:
[[ 1.02363458 -0.13685468 -0.00748725 0.  ]
[-0.13685468 1.15786316 0.07216005 0.  ]
[-0.00748725 0.07216005 1.03830504 0.  ]
[ 0.  0.  0.  1.  ]]
```

The matrix A shows that the gaussian atomic orbital basis set does not form a complete ortho-normal basis set since it would have been the identity if this was the case. Indeed if $S = \mathbb{I}$ then $A = \mathbb{I}$

Exercise 12
What do you observe about the elements of `A`? Is the matrix real or complex? Is the matrix symmetric or not?

Elements of A alsmost all non-zero and are positive and negative-valued. A inherits its properties from the ones of The overlap matrix S which is hermitian ($S = S^\dagger$). Now $A = S^{-1/2} = \sum_{i=1}^{7} \lambda_i^{-1/2} |\lambda_i\rangle\langle\lambda_i|$, where $\{|\lambda_i\rangle\}$ are the eigenvalues of S, so we know that A is hermitian and therefore symmetric as shown in the figure. In our case, A is symmetric and real.

```
array([[ 1.02363458e+00, -1.36854677e-01, -7.48725467e-03,
         0.00000000e+00, -1.08457284e-18,  1.90278847e-02,
         1.90278847e-02],
       [-1.36854677e-01,  1.15786316e+00,  7.21600498e-02,
         0.00000000e+00,  3.64651649e-17, -2.22332521e-01,
        -2.22332521e-01],
       [-7.48725467e-03,  7.21600498e-02,  1.03830504e+00,
         0.00000000e+00, -3.75981753e-17, -1.18462545e-01,
        -1.18462545e-01],
       [ 0.00000000e+00,  0.00000000e+00,  0.00000000e+00,
         1.00000000e+00,  0.00000000e+00,  0.00000000e+00,
         0.00000000e+00],
       [-3.45874404e-19,  5.02534544e-17, -4.01081341e-17,
         0.00000000e+00,  1.07331473e+00,  1.75758279e-01,
        -1.75758279e-01],
       [ 1.90278847e-02, -2.22332521e-01, -1.18462545e-01,
         0.00000000e+00,  1.75758279e-01,  1.12972338e+00,
        -6.25974978e-02],
       [ 1.90278847e-02, -2.22332521e-01, -1.18462545e-01,
         0.00000000e+00, -1.75758279e-01, -6.25974978e-02,
         1.12972338e+00]])
```

Figure 1: Matrix A

Exercise 13
Use the orthogonalization matrix `A` to transform the overlap matrix, `S`. Check the transformed overlap matrix, `S_p`, to make sure it represents an orthonormal basis.

```
# Transform S with A and assign the result to S_p

S_p = A.dot(S.dot(A)) #<your equation here>

isBasisOrthonormal(S_p)
```

```
Q:(T/F) The AO basis is orthonormal?  A: True
```
`True` S' now represents an orthonormal basis.

Exercise 14
The product A S A does not take the complex conjugate transpose of A. What conditions (properties of A) make that ok?

A is hermitian and real, making it a symmetric matrix, indeed we have : $A^\dagger = A$.

Exercise 15
Based on the definition of C', propose a definition of C in terms of A and C'. Justify your equation.

$$C' = A^{-1}C$$
$$AC' = AA^{-1}C$$
$$C = AC' \quad \text{where we used that } A \text{ is invertible such that } A^{-1}A = \mathbb{I}$$

Exercise 16
In the cell below, use the core Hamiltonian matrix as your initial guess for the Fock matrix. Transform it with the same A matrix you used above. To calculate the eigenvalues, `vals`, and eigenvectors, `vecs`, of matrix `M` using `vals, vecs = np.linalg.eigh(M)`.

```
# Guess for the Fock matrix
F = H # Replace

# Transformed Fock matrix
F_p =  A.dot(F.dot(A)) # Replace

# Diagonalize F_p for eigenvalues & eigenvectors with NumPy
e, C_p =  np.linalg.eigh(F_p) # Replace
```

Exercise 17
Display, i.e., `print`, the coefficent matrix and confirm it the correct size

```
print(C_p)
print("The size of the coefficient matrix is:",C_p.shape)
```

```
[[-9.93397459e-01  1.06813566e-01 -1.35307498e-16 -4.17010426e-02
   1.76149298e-15 -3.65697236e-03  1.33010607e-16]
 [-1.14180122e-01 -9.13965568e-01  1.27788258e-15  3.67704764e-01
  -1.55859911e-14  1.28152437e-01 -2.37455910e-15]
 [-7.66151195e-04 -3.68538378e-01 -2.99255600e-15 -9.28905590e-01
   4.00351377e-14  3.62392157e-02 -1.59671447e-15]
 [ 0.00000000e+00 -3.33066907e-16  3.57353036e-16 -4.30211422e-14
  -1.00000000e+00 -3.33066907e-16  5.55111512e-17]
 [ 2.31531739e-18  4.71826794e-17  9.98991226e-01 -3.20984454e-15
   4.77769914e-16 -1.24429917e-15 -4.49058022e-02]
 [-7.86768545e-03 -9.33736227e-02 -3.17531973e-02  9.71167846e-03
  -3.92055707e-16 -7.00803217e-01 -7.06393470e-01]
 [-7.86768545e-03 -9.33736227e-02  3.17531973e-02  9.71167846e-03
   2.74078108e-16 -7.00803217e-01  7.06393470e-01]]
The size of the coefficient matrix is: (7, 7)
```

Exercise 18
Use `A` and the formula you proposed previously to transform the coefficient matrix back to the AO basis. Confirm that the resulting matrix appears reasonable, i.e., similar size and magnitude

```
# Transform the coefficient matrix back into AO basis
# Transform the coefficient matrix back into AO basis
C = A.dot(C_p)
print(C)
```

```
print("\nThe size of C is :",C.shape)
print("Frobenius norm of C to check magnitude : ||C||_F = Tr[C^T C]
    =", np.trace((C.T).dot(C)))
```

```
[[-1.00154358e+00  2.33624458e-01 -3.02733347e-16 -8.56842079e-02
   3.63414278e-15 -4.82226017e-02  9.45116473e-16]
 [ 7.18933084e-03 -1.05793876e+00  1.40864302e-15  3.60110465e-01
  -1.53723457e-14  4.63121174e-01 -9.15741996e-15]
 [ 2.67129883e-04 -4.27284243e-01 -3.00096682e-15 -9.39942475e-01
   4.04447865e-14  2.12940092e-01 -5.11420285e-15]
 [ 0.00000000e+00 -3.33066907e-16  3.57353036e-16 -4.30211422e-14
  -1.00000000e+00 -3.33066907e-16  5.55111512e-17]
 [-2.15774773e-17  6.88978035e-17  1.06107023e+00 -3.15567076e-15
   3.95718955e-16 -6.17464044e-15 -2.96507060e-01]
 [-1.82134850e-03  1.49253289e-01  1.37720978e-01  3.78578935e-02
  -1.61997318e-15 -7.80700283e-01 -8.50140251e-01]
 [-1.82134850e-03  1.49253289e-01 -1.37720978e-01  3.78578935e-02
  -9.93671961e-16 -7.80700283e-01  8.50140251e-01]]

The size of C is : (7, 7)
Frobenius norm of C to check magnitude : ||C||_F = Tr[C^† C] = 2.933564513151939
```

Exercise 19
Build the density matrix, `D`, from the occupied orbitals, `C_occ`, using the function `np.einsum()`. **Hint** Look at (**??**)

```
# Build density matrix from occupied orbitals
D = np.array(np.einsum('pi,qi->pq',C_occ.conj(),C_occ)) # replace
    this command with np.einsum() as described above
print(F'The shape of D is {D.shape}')
```

```
The shape of D is (7, 7)
```

Exercise 20
Define J in terms of the density matrix, `D`, and the electron repulsion integral tensor, `I`, using `np.einsum()`.

```
#Define J
J = np.einsum('rs,pqrs->pq',D,I) # use np.einsum here
```

Exercise 21
Define K in terms of the density matrix, `D`, and the electron repulsion integral tensor, `I`, using `einsum()`.

```
#Define K
K = np.einsum('rq,pqrs->ps',D,I) # use np.einsum here
```

Exercise 22
Define F in terms of H, J, and K. (Recall The Hartree-Fock procedure)

```
#Define F as a function of H, J, and K
F = H + 2*J - K # insert code here
```

Exercise 23
Calculate the SCF energy based on H, F, and D using `np.einsum()`.

```
E_nuc = mol.nuclear_repulsion_energy()
SCF_E = E_nuc + np.einsum('pq,pq->',H + F,D) # insert your code
    here, don't forget to use E_nuc
print(F'Energy = {SCF_E:.8f}')
```

```
Energy = -73.28579644
```

Exercise 24
Based on the result of the calculation in **??**, is this a reasonable answer?

We foud previously :
`The Hartree-Fock ground state energy of the water is: -74.94207989868096 Eh`
The relative error is : $e = \frac{-73.285780+74.94208}{74.94208} \equiv 2.21\%$, which is significant. This means that our initial guess of the Fock matrix being the hamiltonian H leads to an SCF energy that is not a reasonable answer.

Exercise 25
Describe a procedure (i.e. identify the steps) that will update coefficients and compute a new density matrix based on the updated values of the Fock matrix.

We obtained the new Fock matrix $F = H+2J-K$. We should now take $F' = AFA$, find its eigenvalues and eigenvectors which give the coefficient matrix C' and then $C = AC'$ which are the coefficient that we want to optimize in order to minimize SCF Energy. From that we can reavaluate the important quantites such as the electron density matrix D which is now refined compared to before and recalculate SCF energy to check for convergence. Each iteration, the energy decreases by taking the last F since the new one incororates better electron interactions.

Exercise 26
Using the procedure proposed above, calculate the updated coefficients

```
# ==> Nuclear Repulsion Energy <==
E_nuc = mol.nuclear_repulsion_energy()

# ==> SCF Iterations <==
```

```
# Pre-iteration energy declarations
SCF_E = 0.0
E_old = 0.0

# ==> Set default program options <==
# We continue recalculating the energy until it converges to the level we specify.
# The varible ‘E_conv‘ is where we set this level of convergence. We also set a
# maximum number of iterations so that if our calculation does not converge, it
# eventually stops and lets us know that it did not converge.
# Maximum SCF iterations
MAXITER = 40
# Energy convergence criterion
E_conv = 1.0e-6

print('==> Starting SCF Iterations <==\n')

# Begin Iterations
for scf_iter in range(1, MAXITER + 1):

    # Build Fock matrix (meaning define J, K with density matrix and then define F with J and k)

    # <your code here>
    J = np.einsum('rs,pqrs->pq',D.conj(),I)
    K = np.einsum('rq,pqrs->ps',D.conj(),I)
    F = H + 2*J - K


    # Compute SCF energy (don't forget E_nuc!)

    SCF_E = E_nuc + np.einsum('pq,pq->',H + F,D) #<your formula here>

    print(F'SCF Iteration {scf_iter}: Energy = {SCF_E:.8f} dE = {SCF_E - E_old:.8f}')

    # Check to see if the energy is converged.  If it is break out of the loop.
    # If it is not, set the current energy E_old

    if (abs(SCF_E - E_old) < E_conv):
        break
    E_old = SCF_E

    # Compute new coefficient & density matrices (Exercise 26)

    # <your code here>
    F_p = A.dot(F.dot(A))

    # find e and C_p
    e, C_p = np.linalg.eigh(F_p)

    # find C
    C = A.dot(C_p)
```

```
55
56      # Select the occupied MOs
57      C_occ = C[:, :ndocc]
58
59      # Update the density matrix
60      D = np.einsum('pi,qi->pq', C_occ.conj(), C_occ)
61
62      # MAXITER exceeded?
63      if (scf_iter == MAXITER):
64          psi4.core.clean()
65          raise Exception("Maximum number of SCF iterations exceeded.
        ")
66
67  # Post iterations
68  print('\nSCF converged.')
69  print(F'Final RHF Energy: {SCF_E:.6f} [Eh]')
```

```
==> Starting SCF Iterations <==

SCF Iteration 1: Energy = -74.82812526 dE = -74.82812526
SCF Iteration 2: Energy = -74.93548796 dE = -0.10736270
SCF Iteration 3: Energy = -74.94147771 dE = -0.00598975
SCF Iteration 4: Energy = -74.94197197 dE = -0.00049425
SCF Iteration 5: Energy = -74.94205603 dE = -0.00008407
SCF Iteration 6: Energy = -74.94207439 dE = -0.00001836
SCF Iteration 7: Energy = -74.94207862 dE = -0.00000423
SCF Iteration 8: Energy = -74.94207960 dE = -0.00000098

SCF converged.
Final RHF Energy: -74.942080 [Eh]
```

```
1  # ==> Compare our SCF to Psi4 <==
2  # Call psi4.energy() to compute the SCF energy
3  SCF_E_psi = psi4.energy('SCF')
4  psi4.core.clean()
5
6  print(f"PSI4 {SCF_E_psi:.6f}  Ours {SCF_E:.6f}   Absolute
       difference {np.abs(SCF_E-SCF_E_psi)}")
```

```
PSI4 -74.942080 Ours -74.942080 Absolute difference 2.9805295298501733e-07
```

Bonus Exercise 27
Modify the value of E_conv and describe its effect the number of iterations.Provide your code, the output of the SCF calculation and the number of iterations.

```
1  E_conv_values = [1e-3, 5e-4, 1e-4, 5e-5, 1e-6, 5e-7, 1e-8, 5e-9, 1e
       -10, 1e-12]
2
```

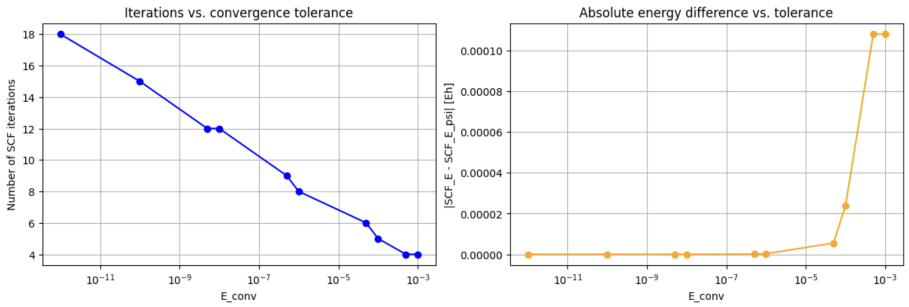
```
iterations_per_econv = []
abs_diffs_per_econv = []

for E_conv in E_conv_values:
    print(f"\n\n===== Running SCF with E_conv = {E_conv:.1e} =====\
    n")

    # We restart the whole process
    # Guess for the Fock matrix
    F = H # Replace

    # Transformed Fock matrix
    F_p =  A.dot(F.dot(A)) # Replace

    # Diagonalize F_p for eigenvalues & eigenvectors with NumPy
    e, C_p =  np.linalg.eigh(F_p) # Replace

    # Retrieve C
    C = A.dot(C_p)

    C_occ = C[:, :ndocc]

    # Build density matrix from occupied orbitals
    D = np.array(np.einsum('pi,qi->pq',C_occ.conj(),C_occ))

    # Build electron repulsion integral (ERI) Tensor
    I = np.asarray(mints.ao_eri())

    #Define J
    J = np.einsum('rs,pqrs->pq',D.conj(),I) # use np.einsum here

    #Define K
    K = np.einsum('rq,pqrs->ps',D.conj(),I) # use np.einsum here

    #Define F as a function of H, J, and K
    F = H + 2*J - K # insert code here

    E_nuc = mol.nuclear_repulsion_energy()
    SCF_E = E_nuc + np.einsum('pq,pq->',H + F,D)

    # insert remaining steps here, you should end with a new C
    # transform F to the orthonormal basis
    F_p = A.dot(F.dot(A))

    # find e and C_p
    e, C_p = np.linalg.eigh(F_p)

    # find C
    C = A.dot(C_p)

    # Select the occupied MOs
    C_occ = C[:, :ndocc]


    # Update the density matrix
    D = np.einsum('pi,qi->pq', C_occ.conj(), C_occ)

```

```
    # ==> Nuclear Repulsion Energy <==
    E_nuc = mol.nuclear_repulsion_energy()

    # ==> SCF Iterations <==
    # Pre-iteration energy declarations
    SCF_E = 0.0
    E_old = 0.0

    # ==> Set default program options <==
    # Maximum SCF iterations
    MAXITER = 40
    # Energy convergence criterion comes from the loop variable
    E_conv

    print('==> Starting SCF Iterations <==\n')

    # Begin Iterations
    for scf_iter in range(1, MAXITER + 1):

        # Build Fock matrix
        J = np.einsum('rs,pqrs->pq',D.conj(),I)
        K = np.einsum('rq,pqrs->ps',D.conj(),I)
        F = H + 2*J - K

        # Compute SCF energy (don't forget E_nuc!)
        SCF_E = E_nuc + np.einsum('pq,pq->',H + F,D)

        print(F'SCF Iteration {scf_iter}: Energy = {SCF_E:.8f} dE =
     {SCF_E - E_old:.8f}')

        # Convergence check
        if (abs(SCF_E - E_old) < E_conv):
            iter_count = scf_iter  # record how many iterations
    this run took
            break
        E_old = SCF_E

        # New coefficients & density
        F_p = A.dot(F.dot(A))
        e, C_p = np.linalg.eigh(F_p)
        C = A.dot(C_p)
        C_occ = C[:, :ndocc]
        D = np.einsum('pi,qi->pq', C_occ.conj(), C_occ)

        # MAXITER exceeded?
        if (scf_iter == MAXITER):
            psi4.core.clean()
            raise Exception("Maximum number of SCF iterations
    exceeded.")

    # Post iterations
    print('\nSCF converged.')
    print(F'Final RHF Energy: {SCF_E:.6f} [Eh]')

    # ==> Compare our SCF to Psi4 <==
    SCF_E_psi = psi4.energy('SCF')
    psi4.core.clean()
```

```
    abs_diff = np.abs(SCF_E - SCF_E_psi)
    print(f"PSI4 {SCF_E_psi:.6f}  Ours {SCF_E:.6f}   Absolute
    difference {abs_diff}")

    # Save metrics for this tolerance
    iterations_per_econv.append(iter_count)
    abs_diffs_per_econv.append(abs_diff)

# === Plots: side-by-side ===
fig, axes = plt.subplots(1, 2, figsize=(12, 4), constrained_layout=
    True)

# Left: iterations vs E_conv
axes[0].plot(E_conv_values, iterations_per_econv, marker='o', color
    ="b")
axes[0].set_xscale('log')
axes[0].set_xlabel('E_conv')
axes[0].set_ylabel('Number of SCF iterations')
axes[0].set_title('Iterations vs. convergence tolerance')
axes[0].grid(True)

# Right: |SCF_E - SCF_E_psi| vs E_conv
axes[1].plot(E_conv_values, abs_diffs_per_econv, marker='o',color =
    "orange")
axes[1].set_xscale('log')
axes[1].set_xlabel('E_conv')
axes[1].set_ylabel('|SCF_E - SCF_E_psi| [Eh]')
axes[1].set_title('Absolute energy difference vs. tolerance')
axes[1].grid(True)

plt.show()
```

Here is the track of the SCF iterations for the following values :

```
E_conv_values = [1e-3, 5e-4, 1e-4, 5e-5, 1e-6, 5e-7, 1e-8, 5e-9, 1e
    -10, 1e-12]
```



We observe that decreasing the threshold E_{conv} leads to higher accuracy (min absolute error of $3.13 \cdot 10^{-13}$) and longuer computation time (18 iterations). Therefore E_{conv} plays a substantial role in terms of accuracy and convergence time, we have to make a trade-off between the two. For such small molecules we don't have much problems but for more complicated molecules we can reasonably extend this notion of trade-off.

```
===== Running SCF with E_conv = 1.0e-03 =====

==> Starting SCF Iterations <==

SCF Iteration 1: Energy = -74.82812526 dE = -74.82812526
SCF Iteration 2: Energy = -74.93548796 dE = -0.10736270
SCF Iteration 3: Energy = -74.94147771 dE = -0.00598975
SCF Iteration 4: Energy = -74.94197197 dE = -0.00049425

SCF converged.
Final RHF Energy: -74.941972 [Eh]
PSI4 -74.942080  Ours -74.941972   Absolute difference 0.00010793216969773312


===== Running SCF with E_conv = 5.0e-04 =====

==> Starting SCF Iterations <==

SCF Iteration 1: Energy = -74.82812526 dE = -74.82812526
SCF Iteration 2: Energy = -74.93548796 dE = -0.10736270
SCF Iteration 3: Energy = -74.94147771 dE = -0.00598975
SCF Iteration 4: Energy = -74.94197197 dE = -0.00049425

SCF converged.
Final RHF Energy: -74.941972 [Eh]
PSI4 -74.942080  Ours -74.941972   Absolute difference 0.00010793216969773312


===== Running SCF with E_conv = 1.0e-04 =====

==> Starting SCF Iterations <==

SCF Iteration 1: Energy = -74.82812526 dE = -74.82812526
SCF Iteration 2: Energy = -74.93548796 dE = -0.10736270
SCF Iteration 3: Energy = -74.94147771 dE = -0.00598975
SCF Iteration 4: Energy = -74.94197197 dE = -0.00049425
SCF Iteration 5: Energy = -74.94205603 dE = -0.00008407

SCF converged.
Final RHF Energy: -74.942056 [Eh]
PSI4 -74.942080  Ours -74.942056   Absolute difference 2.3866800034966218e-05
```

```
===== Running SCF with E_conv = 5.0e-05 =====

==> Starting SCF Iterations <==

SCF Iteration 1: Energy = -74.82812526 dE = -74.82812526
SCF Iteration 2: Energy = -74.93548796 dE = -0.10736270
SCF Iteration 3: Energy = -74.94147771 dE = -0.00598975
SCF Iteration 4: Energy = -74.94197197 dE = -0.00049425
SCF Iteration 5: Energy = -74.94205603 dE = -0.00008407
SCF Iteration 6: Energy = -74.94207439 dE = -0.00001836

SCF converged.
Final RHF Energy: -74.942074 [Eh]
PSI4 -74.942080  Ours -74.942074   Absolute difference 5.509059405994776e-06


===== Running SCF with E_conv = 1.0e-06 =====

==> Starting SCF Iterations <==

SCF Iteration 1: Energy = -74.82812526 dE = -74.82812526
SCF Iteration 2: Energy = -74.93548796 dE = -0.10736270
SCF Iteration 3: Energy = -74.94147771 dE = -0.00598975
SCF Iteration 4: Energy = -74.94197197 dE = -0.00049425
SCF Iteration 5: Energy = -74.94205603 dE = -0.00008407
SCF Iteration 6: Energy = -74.94207439 dE = -0.00001836
SCF Iteration 7: Energy = -74.94207862 dE = -0.00000423
SCF Iteration 8: Energy = -74.94207960 dE = -0.00000098

SCF converged.
Final RHF Energy: -74.942080 [Eh]
PSI4 -74.942080  Ours -74.942080   Absolute difference 2.9805295298501733e-07
```

```
===== Running SCF with E_conv = 5.0e-07 =====

==> Starting SCF Iterations <==

SCF Iteration 1: Energy = -74.82812526 dE = -74.82812526
SCF Iteration 2: Energy = -74.93548796 dE = -0.10736270
SCF Iteration 3: Energy = -74.94147771 dE = -0.00598975
SCF Iteration 4: Energy = -74.94197197 dE = -0.00049425
SCF Iteration 5: Energy = -74.94205603 dE = -0.00008407
SCF Iteration 6: Energy = -74.94207439 dE = -0.00001836
SCF Iteration 7: Energy = -74.94207862 dE = -0.00000423
SCF Iteration 8: Energy = -74.94207960 dE = -0.00000098
SCF Iteration 9: Energy = -74.94207983 dE = -0.00000023

SCF converged.
Final RHF Energy: -74.942080 [Eh]
PSI4 -74.942080  Ours -74.942080   Absolute difference 6.938856245142233e-08


===== Running SCF with E_conv = 1.0e-08 =====

==> Starting SCF Iterations <==

SCF Iteration 1: Energy = -74.82812526 dE = -74.82812526
SCF Iteration 2: Energy = -74.93548796 dE = -0.10736270
SCF Iteration 3: Energy = -74.94147771 dE = -0.00598975
SCF Iteration 4: Energy = -74.94197197 dE = -0.00049425
SCF Iteration 5: Energy = -74.94205603 dE = -0.00008407
SCF Iteration 6: Energy = -74.94207439 dE = -0.00001836
SCF Iteration 7: Energy = -74.94207862 dE = -0.00000423
SCF Iteration 8: Energy = -74.94207960 dE = -0.00000098
SCF Iteration 9: Energy = -74.94207983 dE = -0.00000023
SCF Iteration 10: Energy = -74.94207988 dE = -0.00000005
SCF Iteration 11: Energy = -74.94207989 dE = -0.00000001
SCF Iteration 12: Energy = -74.94207990 dE = -0.00000000

SCF converged.
Final RHF Energy: -74.942080 [Eh]
PSI4 -74.942080  Ours -74.942080   Absolute difference 8.757581326790387e-10
```

```
===== Running SCF with E_conv = 5.0e-09 =====

==> Starting SCF Iterations <==

SCF Iteration 1: Energy = -74.82812526 dE = -74.82812526
SCF Iteration 2: Energy = -74.93548796 dE = -0.10736270
SCF Iteration 3: Energy = -74.94147771 dE = -0.00598975
SCF Iteration 4: Energy = -74.94197197 dE = -0.00049425
SCF Iteration 5: Energy = -74.94205603 dE = -0.00008407
SCF Iteration 6: Energy = -74.94207439 dE = -0.00001836
SCF Iteration 7: Energy = -74.94207862 dE = -0.00000423
SCF Iteration 8: Energy = -74.94207960 dE = -0.00000098
SCF Iteration 9: Energy = -74.94207983 dE = -0.00000023
SCF Iteration 10: Energy = -74.94207988 dE = -0.00000005
SCF Iteration 11: Energy = -74.94207989 dE = -0.00000001
SCF Iteration 12: Energy = -74.94207990 dE = -0.00000000

SCF converged.
Final RHF Energy: -74.942080 [Eh]
PSI4 -74.942080  Ours -74.942080   Absolute difference 8.757581326790387e-10


===== Running SCF with E_conv = 1.0e-10 =====

==> Starting SCF Iterations <==

SCF Iteration 1: Energy = -74.82812526 dE = -74.82812526
SCF Iteration 2: Energy = -74.93548796 dE = -0.10736270
SCF Iteration 3: Energy = -74.94147771 dE = -0.00598975
SCF Iteration 4: Energy = -74.94197197 dE = -0.00049425
SCF Iteration 5: Energy = -74.94205603 dE = -0.00008407
SCF Iteration 6: Energy = -74.94207439 dE = -0.00001836
SCF Iteration 7: Energy = -74.94207862 dE = -0.00000423
SCF Iteration 8: Energy = -74.94207960 dE = -0.00000098
SCF Iteration 9: Energy = -74.94207983 dE = -0.00000023
SCF Iteration 10: Energy = -74.94207988 dE = -0.00000005
SCF Iteration 11: Energy = -74.94207989 dE = -0.00000001
SCF Iteration 12: Energy = -74.94207990 dE = -0.00000000
SCF Iteration 13: Energy = -74.94207990 dE = -0.00000000
SCF Iteration 14: Energy = -74.94207990 dE = -0.00000000
SCF Iteration 15: Energy = -74.94207990 dE = -0.00000000

SCF converged.
Final RHF Energy: -74.942080 [Eh]
PSI4 -74.942080  Ours -74.942080   Absolute difference 1.1169731806148775e-11
```

Figure 2: SCF iterations for different values of E_{conv}

```
===== Running SCF with E_conv = 1.0e-12 =====

==> Starting SCF Iterations <==

SCF Iteration 1: Energy = -74.82812526 dE = -74.82812526
SCF Iteration 2: Energy = -74.93548796 dE = -0.10736270
SCF Iteration 3: Energy = -74.94147771 dE = -0.00598975
SCF Iteration 4: Energy = -74.94197197 dE = -0.00049425
SCF Iteration 5: Energy = -74.94205603 dE = -0.00008407
SCF Iteration 6: Energy = -74.94207439 dE = -0.00001836
SCF Iteration 7: Energy = -74.94207862 dE = -0.00000423
SCF Iteration 8: Energy = -74.94207960 dE = -0.00000098
SCF Iteration 9: Energy = -74.94207983 dE = -0.00000023
SCF Iteration 10: Energy = -74.94207988 dE = -0.00000005
SCF Iteration 11: Energy = -74.94207989 dE = -0.00000001
SCF Iteration 12: Energy = -74.94207990 dE = -0.00000000
SCF Iteration 13: Energy = -74.94207990 dE = -0.00000000
SCF Iteration 14: Energy = -74.94207990 dE = -0.00000000
SCF Iteration 15: Energy = -74.94207990 dE = -0.00000000
SCF Iteration 16: Energy = -74.94207990 dE = -0.00000000
SCF Iteration 17: Energy = -74.94207990 dE = -0.00000000
SCF Iteration 18: Energy = -74.94207990 dE = -0.00000000

SCF converged.
Final RHF Energy: -74.942080 [Eh]
PSI4 -74.942080  Ours -74.942080   Absolute difference 3.126388037344441e-13
```