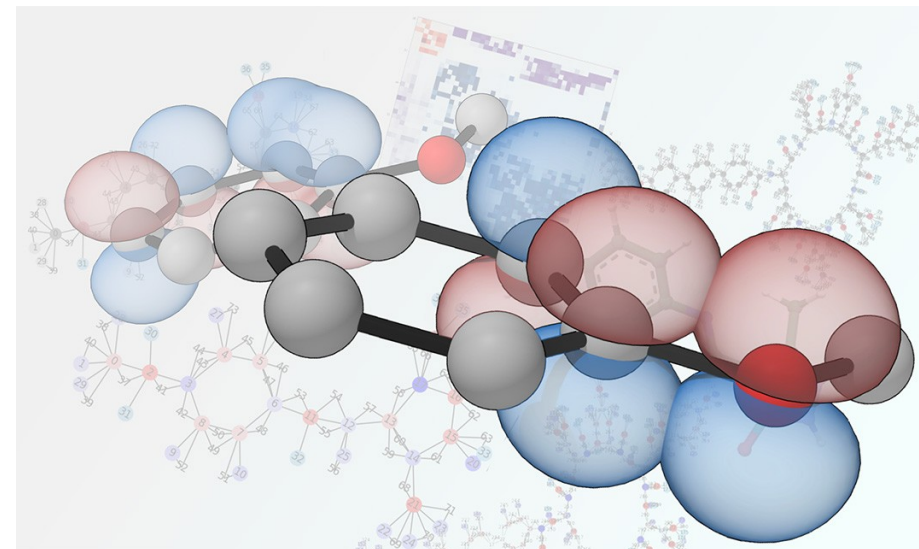


Aula 11 – Gerando Campo de Força

Ref: Cap 10 - Jensen

Prof. Elvis Soares
elvis@peq.coppe.ufrj.br



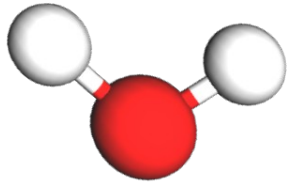
$$i\hbar \frac{\partial}{\partial t} |\Psi\rangle = \hat{H} |\Psi\rangle$$

$$|\Psi\rangle = |\psi\rangle e^{-iEt/\hbar}$$

$$\hat{H} |\psi\rangle = E |\psi\rangle$$

Geração de Campo de Força

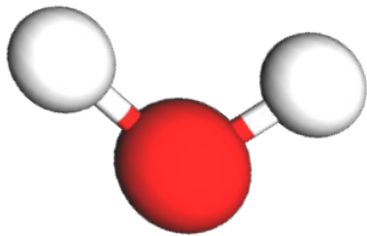
$$E_{\text{FF}} = E_{\text{str}} + E_{\text{bend}} + E_{\text{vdw}} + E_{\text{ele}}$$



Otimizando a Geometria

- **Método:** CCSD(T)
- **Base:** aug-cc-pvdz

	d(Angstrom)	θ (graus)
Exp	0.9578	104.49
CCSD(T)	0.9641	104.13



Definindo a geometria inicial com conjunto de base

```
mol = gto.M(atom = [
    ['O', (0., 0., 0.)],
    ['H', (0., -0.757, -0.587)],
    ['H', (0., 0.757, -0.587)]],
    basis = 'aug-cc-pvdz'
)
```

✓ 0.0s

Definindo a função de otimização da geometria usando método **CCSD(T)**

```
def f(mol):
    # Compute CCSD(T) energy
    mf = scf.RHF(mol).run()
    mycc = cc.CCSD(mf).run()
    et_correction = mycc.ccsd_t()
    e_tot = mycc.e_tot + et_correction

    # Compute CCSD(T) gradients
    g = ccsd_t_grad.Gradients(mycc).kernel()
    print('CCSD(T) nuclear gradients:')
    print(g)
    return e_tot, g
```

```
fake_method = berny_solver.as_pyscf_method(mol, f)
```

✓ 0.0s

Cálculo da otimização da geometria

```
new_mol = berny_solver.optimize(fake_method)
```

🔄 3.2s

Termo de Ligação

```
# modificando o comprimento de ligação O-H para vários valores e calculando as energias CCSD(T)
lengths = np.arange(0.945, 0.985, 0.005)
energies = np.zeros_like(lengths)
for i, length in enumerate(lengths):
    mol_temp = gto.M(atom = f'O 0.000 0.000 0.000;\n\nH 0. {-length*np.sin(0.5*np.radians(angle_HOH))} {length*np.cos(0.5*np.radians(angle_HOH))};\nH 0. {length*np.sin(0.5*np.radians(angle_HOH))} {length*np.cos(0.5*np.radians(angle_HOH))}',\n\nbasis = 'aug-cc-pvdz', verbose = 0)
    mf_temp = scf.RHF(mol_temp).run()
    mycc_temp = cc.CCSD(mf_temp).run()
    et_correction_temp = mycc_temp.ccsd_t()
    e_tot_temp = mycc_temp.e_tot + et_correction_temp
    energies[i] = e_tot_temp
    print(f'Length: {length:.2f} Å, Energy: {e_tot_temp:.6f} Eh')
```

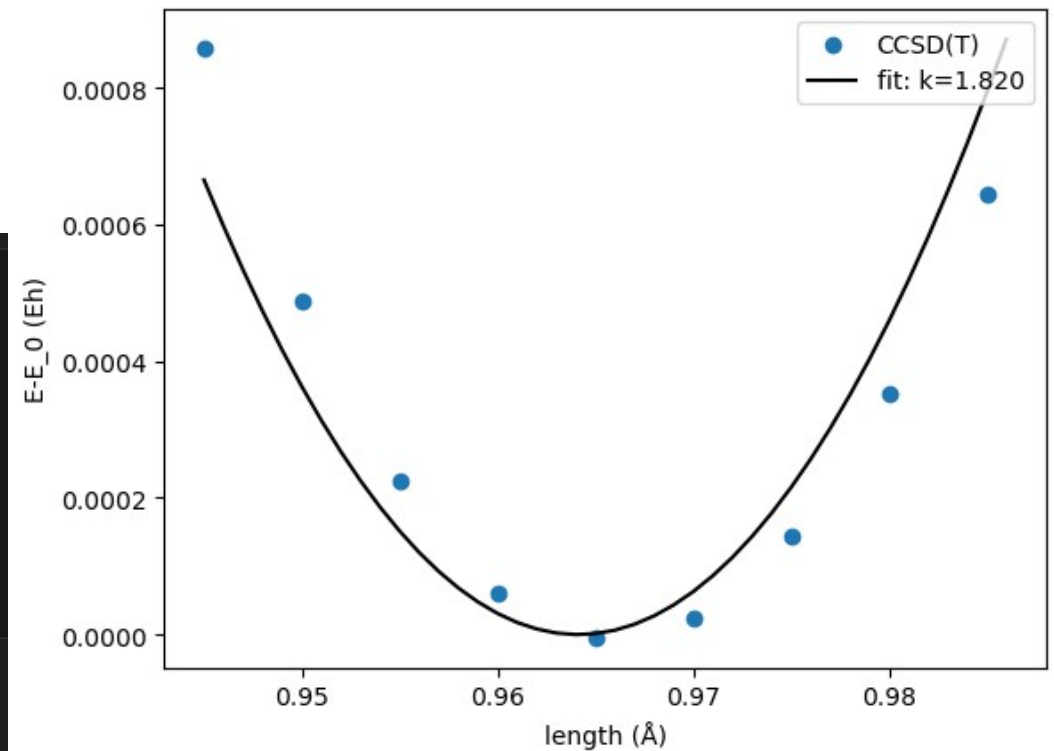
✓ 45.3s

```
Length: 0.94 Å, Energy: -76.275288 Eh
Length: 0.95 Å, Energy: -76.275657 Eh
Length: 0.95 Å, Energy: -76.275921 Eh
Length: 0.96 Å, Energy: -76.276085 Eh
Length: 0.96 Å, Energy: -76.276150 Eh
Length: 0.97 Å, Energy: -76.276121 Eh
Length: 0.97 Å, Energy: -76.276001 Eh
Length: 0.98 Å, Energy: -76.275794 Eh
Length: 0.98 Å, Energy: -76.275501 Eh
```

```
from scipy.optimize import curve_fit

def func(x, k):
    return 2*0.5*k * (x-d_OH1)**2 + e_tot # Cuidado! Aqui temos duas ligações O-H

popt_d, pcov_d = curve_fit(func, lengths, energies, p0=[1.5])
```



$$E_{\text{str}} = E_0 + \frac{1}{2}k_b(l - l_0)^2$$

$$k_b = 1141.91 \text{ kcal}/(\text{mol} \cdot \text{\AA}^2)$$

Termo de ângulo

```
# modificando o ângulo de ligação H-O-H para vários valores e calculando as energias CCSD(T)
angles = np.arange(100, 109, 1.0)
energies = np.zeros_like(angles)
for i, angle in enumerate(angles):
    mol_temp = gto.M(atom = f'O 0.000 0.000 0.000;\n\nH 0. {-d_OH1*np.sin(0.5*np.radians(angle))} {-d_OH1*np.cos(0.5*np.radians(angle))}\n\nH 0. {-d_OH2*np.sin(0.5*np.radians(angle))} {-d_OH2*np.cos(0.5*np.radians(angle))}',
                      basis = 'aug-cc-pvdz', verbose = 0)
    mf_temp = scf.RHF(mol_temp).run()
    mycc_temp = cc.CCSD(mf_temp).run()
    et_correction_temp = mycc_temp.ccsd_t()
    e_tot_temp = mycc_temp.e_tot + et_correction_temp
    energies[i] = e_tot_temp
    print(f'angle: {angle:.2f} degrees, Energy: {e_tot_temp:.6f} Eh')
```

✓ 43.4s

```
angle: 100.00 degrees, Energy: -76.275733 Eh
angle: 101.00 degrees, Energy: -76.275913 Eh
angle: 102.00 degrees, Energy: -76.276042 Eh
angle: 103.00 degrees, Energy: -76.276119 Eh
angle: 104.00 degrees, Energy: -76.276146 Eh
angle: 105.00 degrees, Energy: -76.276123 Eh
angle: 106.00 degrees, Energy: -76.276052 Eh
angle: 107.00 degrees, Energy: -76.275933 Eh
angle: 108.00 degrees, Energy: -76.275768 Eh
```

Gerar

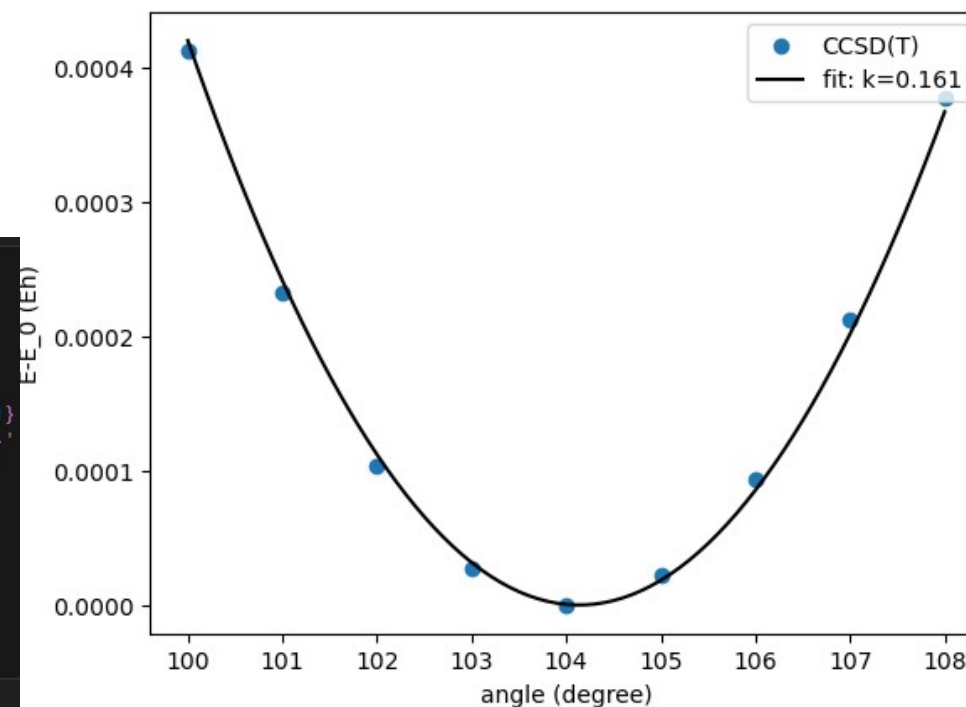
+ Código

+ Markdown

```
from scipy.optimize import curve_fit

def func(x, k):
    return 0.5*k * (np.radians(x)-np.radians(angle_HOH))**2 + e_tot

popt_theta, pcov_theta = curve_fit(func, angles, energies, p0=[0.3])
```



$$E_{\text{bend}} = E_0 + \frac{1}{2}k_{\theta}(\theta - \theta_0)^2$$

$$k_{\theta} = 101.22 \text{ kcal}/(\text{mol}\cdot\text{rad}^2)$$

Termo eletrostático

```
from gpu4pyscf.pop import esp
```

```
# ESP charge
```

```
q0 = esp.esp_solve(new_mol, dm)
```

```
print('Fitted ESP charge')
```

```
print(q0)
```

```
✓ 0.0s Abrir "q0" no Data Wrangler
```

```
Fitted ESP charge
```

```
[-0.72845263  0.36422632  0.36422632]
```

```
# RESP charge // first stage fitting
```

```
q1 = esp.resp_solve(new_mol, dm)
```

```
# Constraint for equal charges
```

```
# equal_constraints = [
```

```
#     [i,j,k],
```

```
#     [u,v,w]]
```

```
#     -->
```

```
#     q[i] = q[j] = q[k] = q[l]
```

```
#     q[u] = q[v] = q[w]
```

```
equal_constraints = [[1,2]] # H atoms in H2O are equal
```

```
# RESP charge // second stage fitting
```

```
q2 = esp.resp_solve(new_mol, dm, resp_a=1e-3,
```

```
                    equal_constraints=equal_constraints)
```

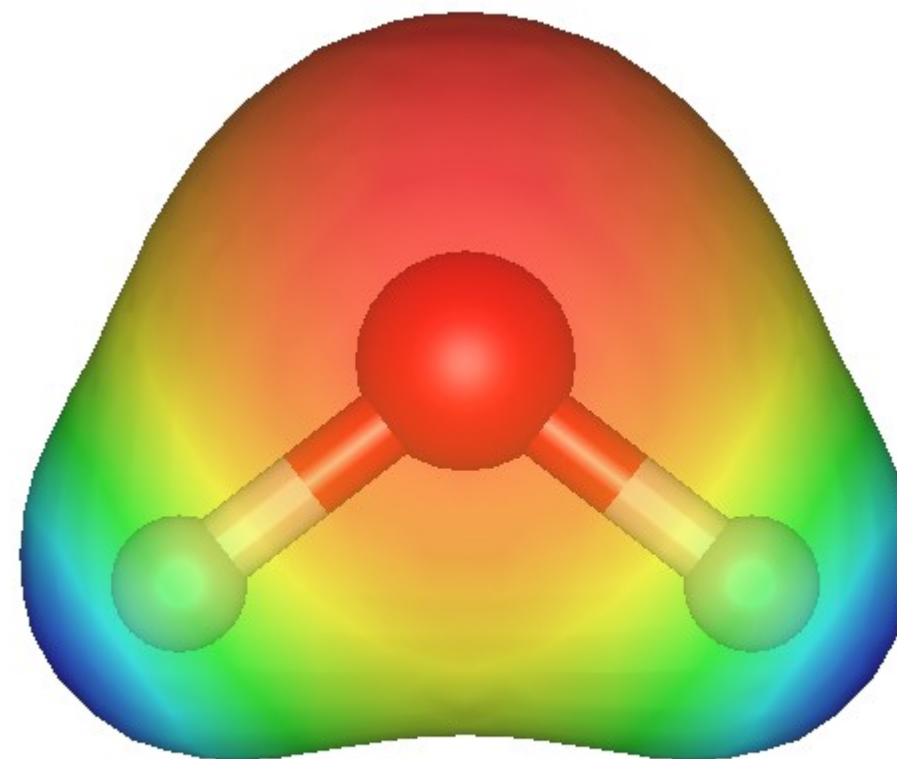
```
print('Fitted partial charge with RESP')
```

```
print(q2)
```

```
✓ 0.0s Abrir "q2" no Data Wrangler
```

```
Fitted partial charge with RESP
```

```
[-0.72445366  0.36222683  0.36222683]
```



$$Q_O = -0.724$$

$$Q_H = 0.362$$

Interação entre 2 moléculas

