Normal Modes

Normal modes

Hessian
- Local curvature of the potential energy
- If in local minimum: vibrational modes

Normal modes
- Mass-weighted Hessian $F_{ij}=H_{ij}(M_iM_j)^{-1/2}$
- Eigenvalues: Frequencies
- Eigenvectors: Modes

$$\mathbf{H} = \begin{bmatrix} \frac{\partial^2 E}{\partial x_1^2} & \frac{\partial^2 E}{\partial x_1\,\partial x_2} & \cdots & \frac{\partial^2 E}{\partial x_1\,\partial x_n} \\ \frac{\partial^2 E}{\partial x_2\,\partial x_1} & \frac{\partial^2 E}{\partial x_2^2} & \cdots & \frac{\partial^2 E}{\partial x_2\,\partial x_n} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial^2 E}{\partial x_n\,\partial x_1} & \frac{\partial^2 E}{\partial x_n\,\partial x_2} & \cdots & \frac{\partial^2 E}{\partial x_n^2} \end{bmatrix},$$

Normal modes


```python
from pyscf import gto
import numpy as np
from pyscf.data import nist

# Build molecule
mol = gto.M(atom=[['O', 0., 0., 0.106817],
                  ['H', 0., -0.785198 , -0.427268],
                  ['H', 0., 0.785198 , -0.427268]],
            basis='6-31G',
            verbose=0)

# Do Hartree-Fock calculation
mf = mol.RHF().run()

# Calculate Hessian
hessian = mf.Hessian().kernel()
# Change the array layout
hessian = hessian.transpose(0, 2, 1, 3).reshape(mol.natm * 3, mol.natm * 3)

# Mass-weight the Hessian
atom_masses = mol.atom_mass_list(isotope_avg=True)
atom_masses = np.repeat(atom_masses, 3)
Mhalf = 1/np.sqrt(np.outer(atom_masses, atom_masses))
weighted_hessian = hessian * Mhalf

# Calculate eigenvalues and eigenvectors
force_constants, modes = np.linalg.eigh(weighted_hessian)

# Change units for wavenumbers
frequencies = np.sqrt(np.abs(force_constants))
to_wavenumbers = (nist.HARTREE2J / (nist.ATOMIC_MASS * nist.BOHR_SI**2))**.5
to_wavenumbers *= 1/(2 * np.pi) / nist.LIGHT_SPEED_SI * 1e-2
to_wavenumbers * np.sort(frequencies)[-3:]
```

```
array([1736.71755055, 3988.23212533, 4145.21010618])
```

Imports

- PySCF: Quantum chemistry calculations
- Numpy: Mathematical operations

Normal modes


```python
from pyscf import gto
import numpy as np
from pyscf.data import nist

# Build molecule
mol = gto.M(atom=[['O', 0., 0., 0.106817],
                  ['H', 0., -0.785198 , -0.427268],
                  ['H', 0., 0.785198 , -0.427268]],
            basis='6-31G',
            verbose=0)

# Do Hartree-Fock calculation
mf = mol.RHF().run()

# Calculate Hessian
hessian = mf.Hessian().kernel()
# Change the array layout
hessian = hessian.transpose(0, 2, 1, 3).reshape(mol.natm * 3, mol.natm * 3)

# Mass-weight the Hessian
atom_masses = mol.atom_mass_list(isotope_avg=True)
atom_masses = np.repeat(atom_masses, 3)
Mhalf = 1/np.sqrt(np.outer(atom_masses, atom_masses))
weighted_hessian = hessian * Mhalf

# Calculate eigenvalues and eigenvectors
force_constants, modes = np.linalg.eigh(weighted_hessian)

# Change units for wavenumbers
frequencies = np.sqrt(np.abs(force_constants))
to_wavenumbers = (nist.HARTREE2J / (nist.ATOMIC_MASS * nist.BOHR_SI**2))**.5
to_wavenumbers *= 1/(2 * np.pi) / nist.LIGHT_SPEED_SI * 1e-2
to_wavenumbers * np.sort(frequencies)[-3:]
```

```
array([1736.71755055, 3988.23212533, 4145.21010618])
```

Define a molecule: water

- Coordinates (need to be a minimum!)
- Basis set: measure of accuracy

Normal modes


```python
from pyscf import gto
import numpy as np
from pyscf.data import nist

# Build molecule
mol = gto.M(atom=[['O', 0., 0., 0.106817],
                  ['H', 0., -0.785198 , -0.427268],
                  ['H', 0., 0.785198 , -0.427268]],
            basis='6-31G',
            verbose=0)

# Do Hartree-Fock calculation
mf = mol.RHF().run()

# Calculate Hessian
hessian = mf.Hessian().kernel()
# Change the array layout
hessian = hessian.transpose(0, 2, 1, 3).reshape(mol.natm * 3, mol.natm * 3)

# Mass-weight the Hessian
atom_masses = mol.atom_mass_list(isotope_avg=True)
atom_masses = np.repeat(atom_masses, 3)
Mhalf = 1/np.sqrt(np.outer(atom_masses, atom_masses))
weighted_hessian = hessian * Mhalf

# Calculate eigenvalues and eigenvectors
force_constants, modes = np.linalg.eigh(weighted_hessian)

# Change units for wavenumbers
frequencies = np.sqrt(np.abs(force_constants))
to_wavenumbers = (nist.HARTREE2J / (nist.ATOMIC_MASS * nist.BOHR_SI**2))**.5
to_wavenumbers *= 1/(2 * np.pi) / nist.LIGHT_SPEED_SI * 1e-2
to_wavenumbers * np.sort(frequencies)[-3:]
```

```
array([1736.71755055, 3988.23212533, 4145.21010618])
```

Do the actual calculation

- Restricted Hartree-Fock as a method
- For large molecules: expensive

Normal modes


```python
from pyscf import gto
import numpy as np
from pyscf.data import nist

# Build molecule
mol = gto.M(atom=[['O', 0., 0., 0.106817],
                  ['H', 0., -0.785198 , -0.427268],
                  ['H', 0., 0.785198 , -0.427268]],
            basis='6-31G',
            verbose=0)

# Do Hartree-Fock calculation
mf = mol.RHF().run()

# Calculate Hessian
hessian = mf.Hessian().kernel()
# Change the array layout
hessian = hessian.transpose(0, 2, 1, 3).reshape(mol.natm * 3, mol.natm * 3)

# Mass-weight the Hessian
atom_masses = mol.atom_mass_list(isotope_avg=True)
atom_masses = np.repeat(atom_masses, 3)
Mhalf = 1/np.sqrt(np.outer(atom_masses, atom_masses))
weighted_hessian = hessian * Mhalf

# Calculate eigenvalues and eigenvectors
force_constants, modes = np.linalg.eigh(weighted_hessian)

# Change units for wavenumbers
frequencies = np.sqrt(np.abs(force_constants))
to_wavenumbers = (nist.HARTREE2J / (nist.ATOMIC_MASS * nist.BOHR_SI**2))**.5
to_wavenumbers *= 1/(2 * np.pi) / nist.LIGHT_SPEED_SI * 1e-2
to_wavenumbers * np.sort(frequencies)[-3:]
```

```
array([1736.71755055, 3988.23212533, 4145.21010618])
```

Calculate the analytical Hessian

- For large molecules: even more expensive

Normal modes


```python
from pyscf import gto
import numpy as np
from pyscf.data import nist

# Build molecule
mol = gto.M(atom=[['O', 0., 0., 0.106817],
                  ['H', 0., -0.785198 , -0.427268],
                  ['H', 0., 0.785198 , -0.427268]],
            basis='6-31G',
            verbose=0)

# Do Hartree-Fock calculation
mf = mol.RHF().run()

# Calculate Hessian
hessian = mf.Hessian().kernel()
# Change the array layout
hessian = hessian.transpose(0, 2, 1, 3).reshape(mol.natm * 3, mol.natm * 3)

# Mass-weight the Hessian
atom_masses = mol.atom_mass_list(isotope_avg=True)
atom_masses = np.repeat(atom_masses, 3)
Mhalf = 1/np.sqrt(np.outer(atom_masses, atom_masses))
weighted_hessian = hessian * Mhalf

# Calculate eigenvalues and eigenvectors
force_constants, modes = np.linalg.eigh(weighted_hessian)

# Change units for wavenumbers
frequencies = np.sqrt(np.abs(force_constants))
to_wavenumbers = (nist.HARTREE2J / (nist.ATOMIC_MASS * nist.BOHR_SI**2))**.5
to_wavenumbers *= 1/(2 * np.pi) / nist.LIGHT_SPEED_SI * 1e-2
to_wavenumbers * np.sort(frequencies)[-3:]
```

```
array([1736.71755055, 3988.23212533, 4145.21010618])
```

Change the memory layout

- PySCF returns the Hessian as four-dimensional array
  (atom1, atom2, dimension1, dimension2)
- We need it as square symmetric matrix

np.transpose()

- Transposes a matrix = changes axes order
- By default: reverse axes
- Here: Sort into
  (atom1, dimension1, atom2, dimension2)

np.reshape()

- Keeps data, looks at it differently
- Here: Makes matrix square

Normal modes


```python
from pyscf import gto
import numpy as np
from pyscf.data import nist

# Build molecule
mol = gto.M(atom=[['O', 0., 0., 0.106817],
                  ['H', 0., -0.785198 , -0.427268],
                  ['H', 0., 0.785198 , -0.427268]],
            basis='6-31G',
            verbose=0)

# Do Hartree-Fock calculation
mf = mol.RHF().run()

# Calculate Hessian
hessian = mf.Hessian().kernel()
# Change the array layout
hessian = hessian.transpose(0, 2, 1, 3).reshape(mol.natm * 3, mol.natm * 3)

# Mass-weight the Hessian
atom_masses = mol.atom_mass_list(isotope_avg=True)
atom_masses = np.repeat(atom_masses, 3)
Mhalf = 1/np.sqrt(np.outer(atom_masses, atom_masses))
weighted_hessian = hessian * Mhalf

# Calculate eigenvalues and eigenvectors
force_constants, modes = np.linalg.eigh(weighted_hessian)

# Change units for wavenumbers
frequencies = np.sqrt(np.abs(force_constants))
to_wavenumbers = (nist.HARTREE2J / (nist.ATOMIC_MASS * nist.BOHR_SI**2))**.5
to_wavenumbers *= 1/(2 * np.pi) / nist.LIGHT_SPEED_SI * 1e-2
to_wavenumbers * np.sort(frequencies)[-3:]
```

```
array([1736.71755055, 3988.23212533, 4145.21010618])
```

Build list of atomic masses

- PySCF has built-in data sets, no copying required

np.repeat()

- Repeats each element
- Once for each dimension

Normal modes


```python
from pyscf import gto
import numpy as np
from pyscf.data import nist

# Build molecule
mol = gto.M(atom=[['O', 0., 0., 0.106817],
                  ['H', 0., -0.785198 , -0.427268],
                  ['H', 0., 0.785198 , -0.427268]],
            basis='6-31G',
            verbose=0)

# Do Hartree-Fock calculation
mf = mol.RHF().run()

# Calculate Hessian
hessian = mf.Hessian().kernel()
# Change the array layout
hessian = hessian.transpose(0, 2, 1, 3).reshape(mol.natm * 3, mol.natm * 3)

# Mass-weight the Hessian
atom_masses = mol.atom_mass_list(isotope_avg=True)
atom_masses = np.repeat(atom_masses, 3)
Mhalf = 1/np.sqrt(np.outer(atom_masses, atom_masses))
weighted_hessian = hessian * Mhalf

# Calculate eigenvalues and eigenvectors
force_constants, modes = np.linalg.eigh(weighted_hessian)

# Change units for wavenumbers
frequencies = np.sqrt(np.abs(force_constants))
to_wavenumbers = (nist.HARTREE2J / (nist.ATOMIC_MASS * nist.BOHR_SI**2))**.5
to_wavenumbers *= 1/(2 * np.pi) / nist.LIGHT_SPEED_SI * 1e-2
to_wavenumbers * np.sort(frequencies)[-3:]
```

```
array([1736.71755055, 3988.23212533, 4145.21010618])
```

Build mass-weighting matrix

- Note that *np.sqrt()* is operating elementwise
- *matrix * matrix* is elementwise (*np.matmul()* would be matrix multiplication)

np.outer()

- Outer product
- Pairwise multiplication: $M_{ij} = m_i m_j$

Normal modes


```python
from pyscf import gto
import numpy as np
from pyscf.data import nist

# Build molecule
mol = gto.M(atom=[['O', 0., 0., 0.106817],
                  ['H', 0., -0.785198 , -0.427268],
                  ['H', 0., 0.785198 , -0.427268]],
            basis='6-31G',
            verbose=0)

# Do Hartree-Fock calculation
mf = mol.RHF().run()

# Calculate Hessian
hessian = mf.Hessian().kernel()
# Change the array layout
hessian = hessian.transpose(0, 2, 1, 3).reshape(mol.natm * 3, mol.natm * 3)

# Mass-weight the Hessian
atom_masses = mol.atom_mass_list(isotope_avg=True)
atom_masses = np.repeat(atom_masses, 3)
Mhalf = 1/np.sqrt(np.outer(atom_masses, atom_masses))
weighted_hessian = hessian * Mhalf

# Calculate eigenvalues and eigenvectors
force_constants, modes = np.linalg.eigh(weighted_hessian)

# Change units for wavenumbers
frequencies = np.sqrt(np.abs(force_constants))
to_wavenumbers = (nist.HARTREE2J / (nist.ATOMIC_MASS * nist.BOHR_SI**2))**.5
to_wavenumbers *= 1/(2 * np.pi) / nist.LIGHT_SPEED_SI * 1e-2
to_wavenumbers * np.sort(frequencies)[-3:]
```

```
array([1736.71755055, 3988.23212533, 4145.21010618])
```

Get eigenvalues

- From weighted Hessian(!)

np.eigh()

- For symmetric matrices
- Otherwise: *np.eig()*

Normal modes


```python
from pyscf import gto
import numpy as np
from pyscf.data import nist

# Build molecule
mol = gto.M(atom=[['O', 0., 0., 0.106817],
                  ['H', 0., -0.785198 , -0.427268],
                  ['H', 0., 0.785198 , -0.427268]],
            basis='6-31G',
            verbose=0)

# Do Hartree-Fock calculation
mf = mol.RHF().run()

# Calculate Hessian
hessian = mf.Hessian().kernel()
# Change the array layout
hessian = hessian.transpose(0, 2, 1, 3).reshape(mol.natm * 3, mol.natm * 3)

# Mass-weight the Hessian
atom_masses = mol.atom_mass_list(isotope_avg=True)
atom_masses = np.repeat(atom_masses, 3)
Mhalf = 1/np.sqrt(np.outer(atom_masses, atom_masses))
weighted_hessian = hessian * Mhalf

# Calculate eigenvalues and eigenvectors
force_constants, modes = np.linalg.eigh(weighted_hessian)

# Change units for wavenumbers
frequencies = np.sqrt(np.abs(force_constants))
to_wavenumbers = (nist.HARTREE2J / (nist.ATOMIC_MASS * nist.BOHR_SI**2))**.5
to_wavenumbers *= 1/(2 * np.pi) / nist.LIGHT_SPEED_SI * 1e-2
to_wavenumbers * np.sort(frequencies)[-3:]
```

```
array([1736.71755055, 3988.23212533, 4145.21010618])
```

Get frequencies from force constants

- Harmonic approximation
- Negative and small entries:
  3 translational and 3 rotational degrees of freedom

np.abs()

- Absolute value
- Here: shortcut (better: remove translational and rotational degrees first)

Normal modes


```python
from pyscf import gto
import numpy as np
from pyscf.data import nist

# Build molecule
mol = gto.M(atom=[['O', 0., 0., 0.106817],
                  ['H', 0., -0.785198 , -0.427268],
                  ['H', 0., 0.785198 , -0.427268]],
            basis='6-31G',
            verbose=0)

# Do Hartree-Fock calculation
mf = mol.RHF().run()

# Calculate Hessian
hessian = mf.Hessian().kernel()
# Change the array layout
hessian = hessian.transpose(0, 2, 1, 3).reshape(mol.natm * 3, mol.natm * 3)

# Mass-weight the Hessian
atom_masses = mol.atom_mass_list(isotope_avg=True)
atom_masses = np.repeat(atom_masses, 3)
Mhalf = 1/np.sqrt(np.outer(atom_masses, atom_masses))
weighted_hessian = hessian * Mhalf

# Calculate eigenvalues and eigenvectors
force_constants, modes = np.linalg.eigh(weighted_hessian)

# Change units for wavenumbers
frequencies = np.sqrt(np.abs(force_constants))
to_wavenumbers = (nist.HARTREE2J / (nist.ATOMIC_MASS * nist.BOHR_SI**2))**.5
to_wavenumbers *= 1/(2 * np.pi) / nist.LIGHT_SPEED_SI * 1e-2
to_wavenumbers * np.sort(frequencies)[-3:]
```

```
array([1736.71755055, 3988.23212533, 4145.21010618])
```

Convert to wavenumbers

- PySCF has predefined constants

PySCF can do it

```python
from pyscf import gto
from pyscf.hessian.thermo import harmonic_analysis

# Build molecule
mol = gto.M(atom=[['O', 0., 0., 0.106817],
                  ['H', 0., -0.785198 , -0.427268],
                  ['H', 0., 0.785198 , -0.427268]],
            basis='6-31G',
            verbose=0)

# Do Hartree-Fock calculation
mf = mol.RHF().run()

# Calculate Hessian
hessian = mf.Hessian().kernel()

harmonic_analysis(mol, hessian)['freq_wavenumber']
```

```
array([1736.71755056, 3988.23212533, 4145.21010587])
```