

Supplemental Information for The Role of High-Order Electron Correlation Effects in a Model System for Non-valence Correlation-bound Anions

Shiv Upadhyay, Amanda Dumi, James Shee, Kenneth D. Jordan

September 24, 2020

CONTENTS

| | |
|--|-----------|
| 1 Supplemental Basis Functions | 2 |
| 1.a 7s7p | 2 |
| 1.b 3s1p | 3 |
| 2 DMC extrapolation | 4 |
| 3 Radial orbital density plots | 14 |
| 3.a Required software versions | 15 |
| 3.b Step 1: Generating a Molden file | 15 |
| 3.c Step 2: Integrating over the angular components of the singly occupied orbital | 15 |

1 SUPPLEMENTAL BASIS FUNCTIONS

The exponents for the 7s7p and 3s1p diffuse Gaussian type orbitals are given below in the GAMESS format.

1.a 7s7p

```
S    1
 1   0.02362232   1.0
S    1
 1   0.00738198   1.0
S    1
 1   0.00230687   1.0
S    1
 1   0.00072090   1.0
S    1
 1   0.00022528   1.0
S    1
 1   0.00007040   1.0
P    1
 1   0.02362232   1.0
P    1
 1   0.00738198   1.0
P    1
 1   0.00230687   1.0
P    1
```

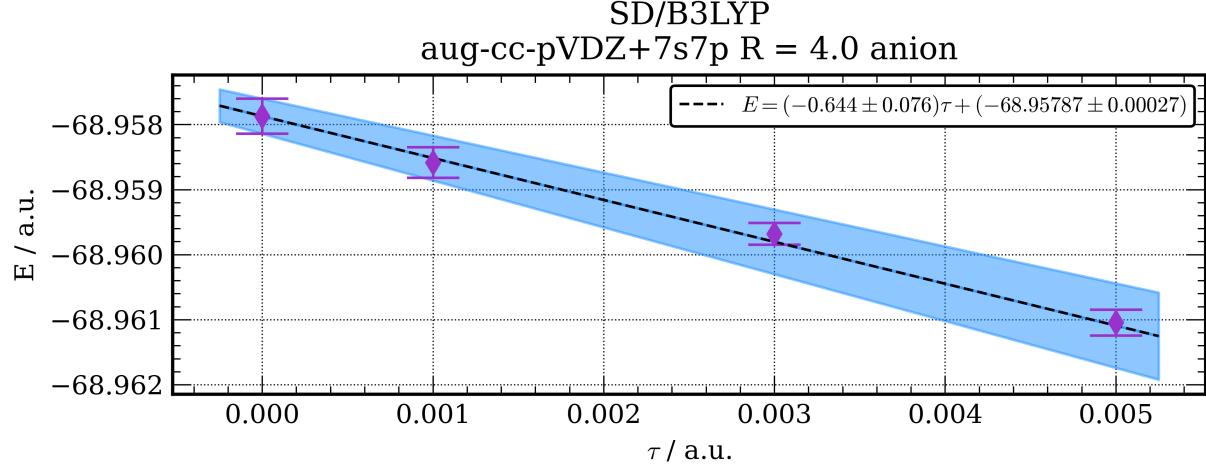
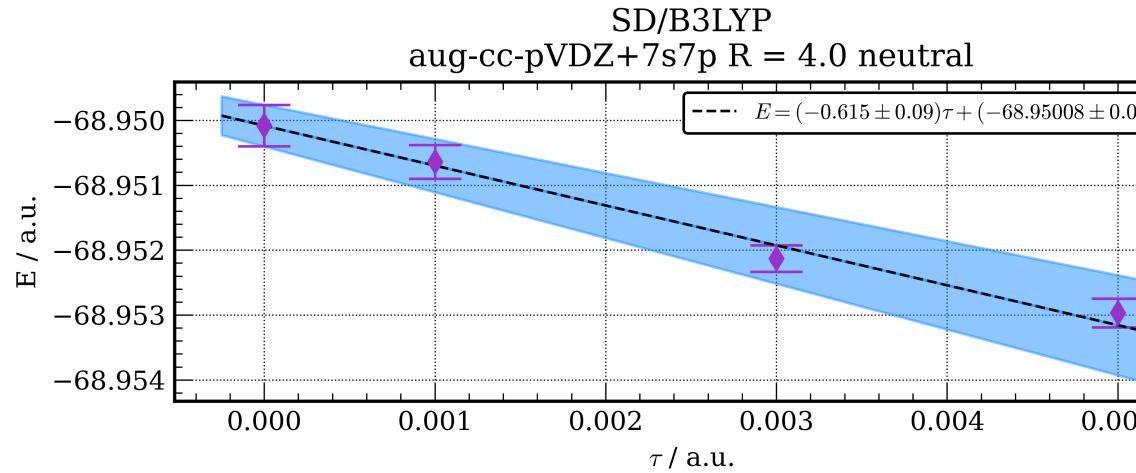
| | | |
|---|------------|-----|
| 1 | 0.00072090 | 1.0 |
| P | 1 | |
| 1 | 0.00022528 | 1.0 |
| P | 1 | |
| 1 | 0.00007040 | 1.0 |
| P | 1 | |
| 1 | 0.00002200 | 1.0 |

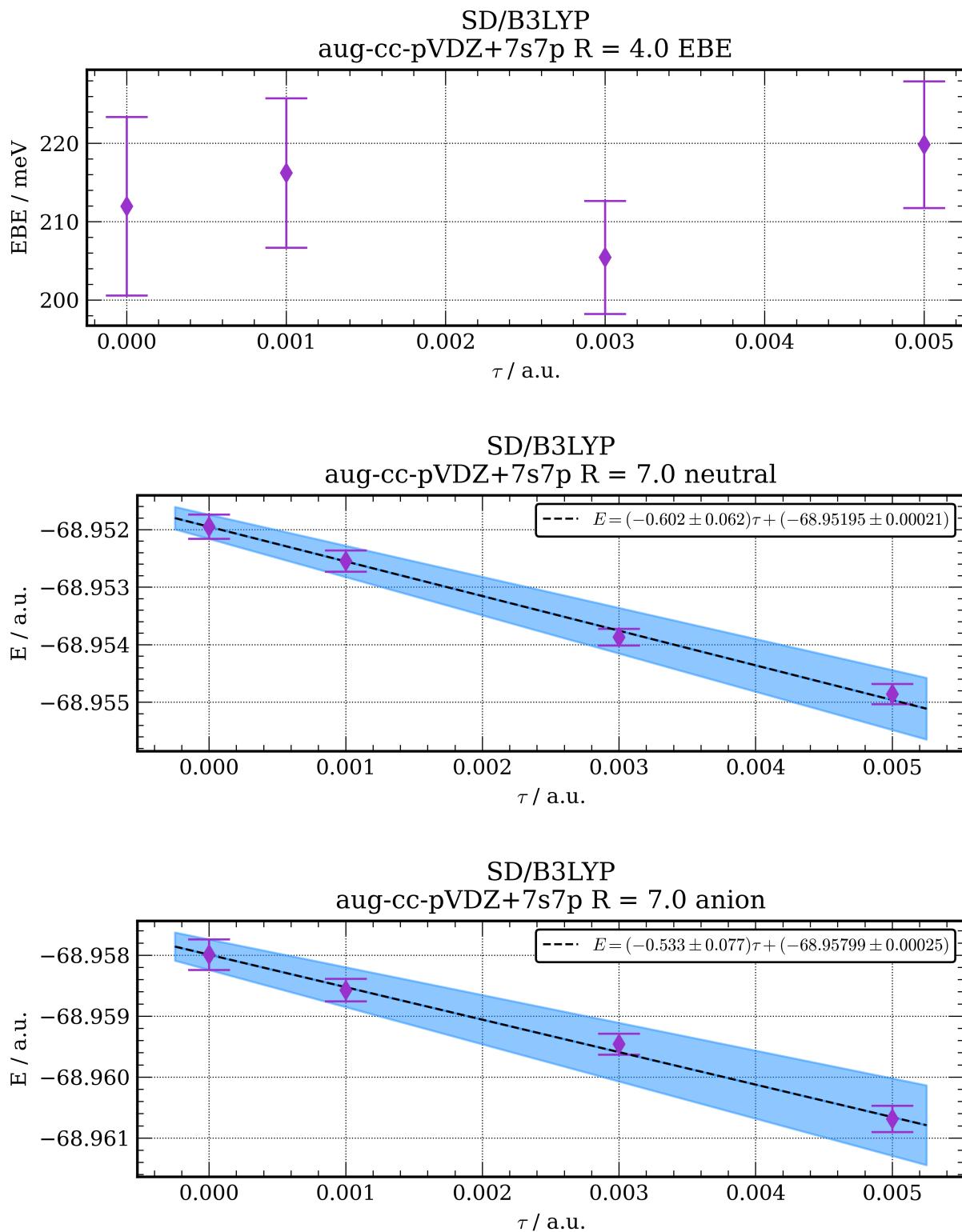
1.b 3s1p

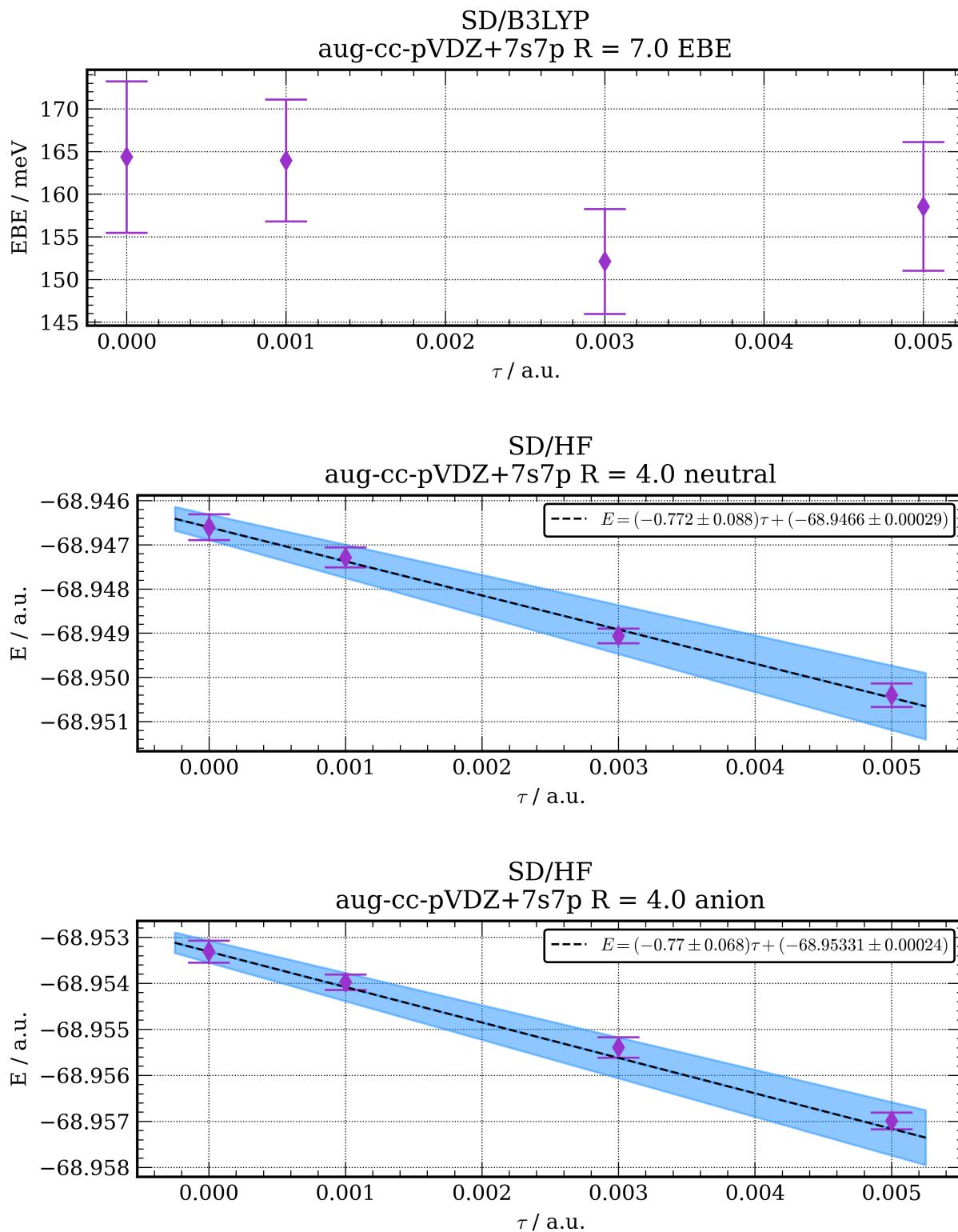
| | | |
|---|------------|-----|
| S | 1 | |
| 1 | 0.02362232 | 1.0 |
| S | 1 | |
| 1 | 0.00738198 | 1.0 |
| S | 1 | |
| 1 | 0.00230687 | 1.0 |
| P | 1 | |
| 1 | 0.02362232 | 1.0 |

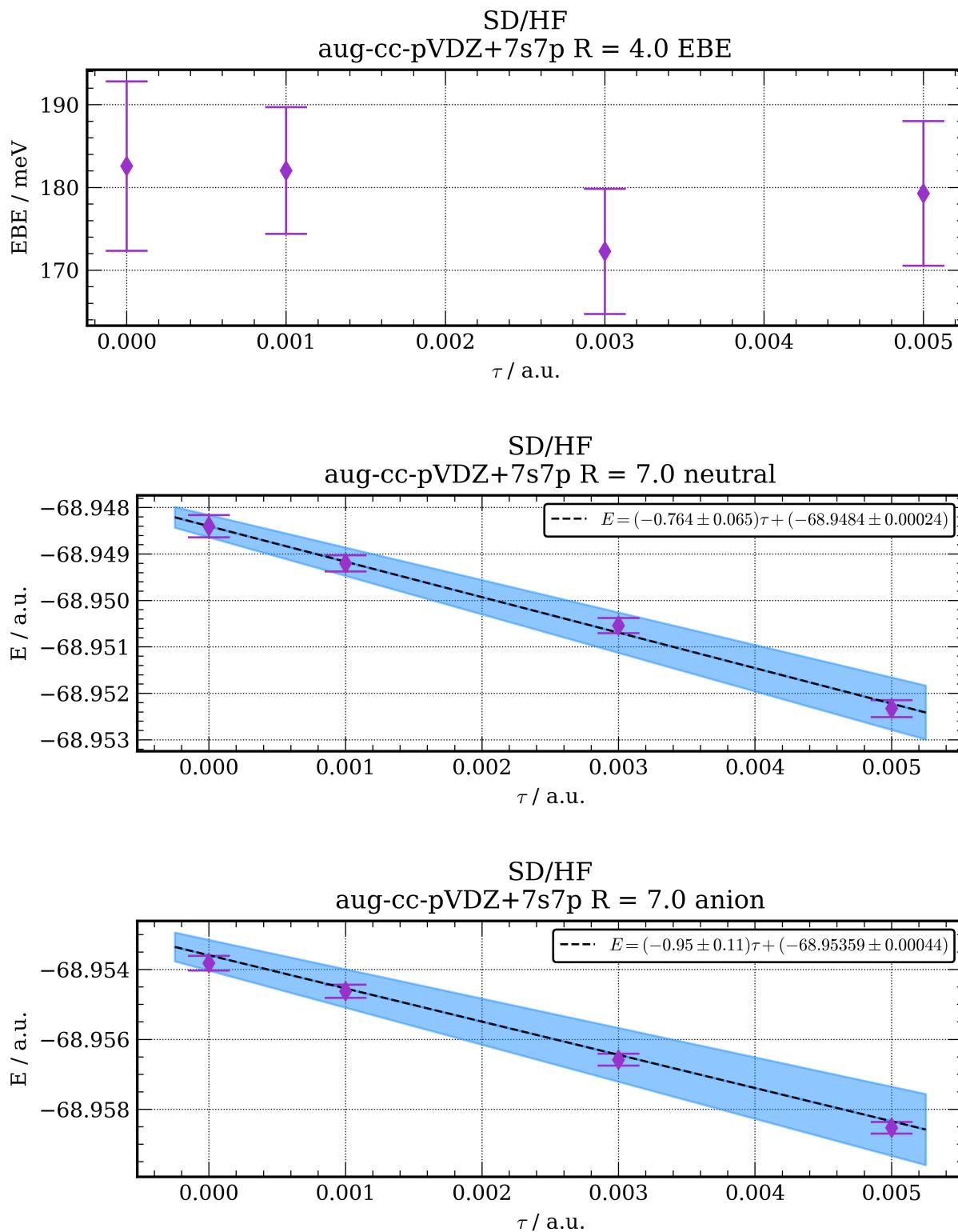
2 DMC EXTRAPOLATION

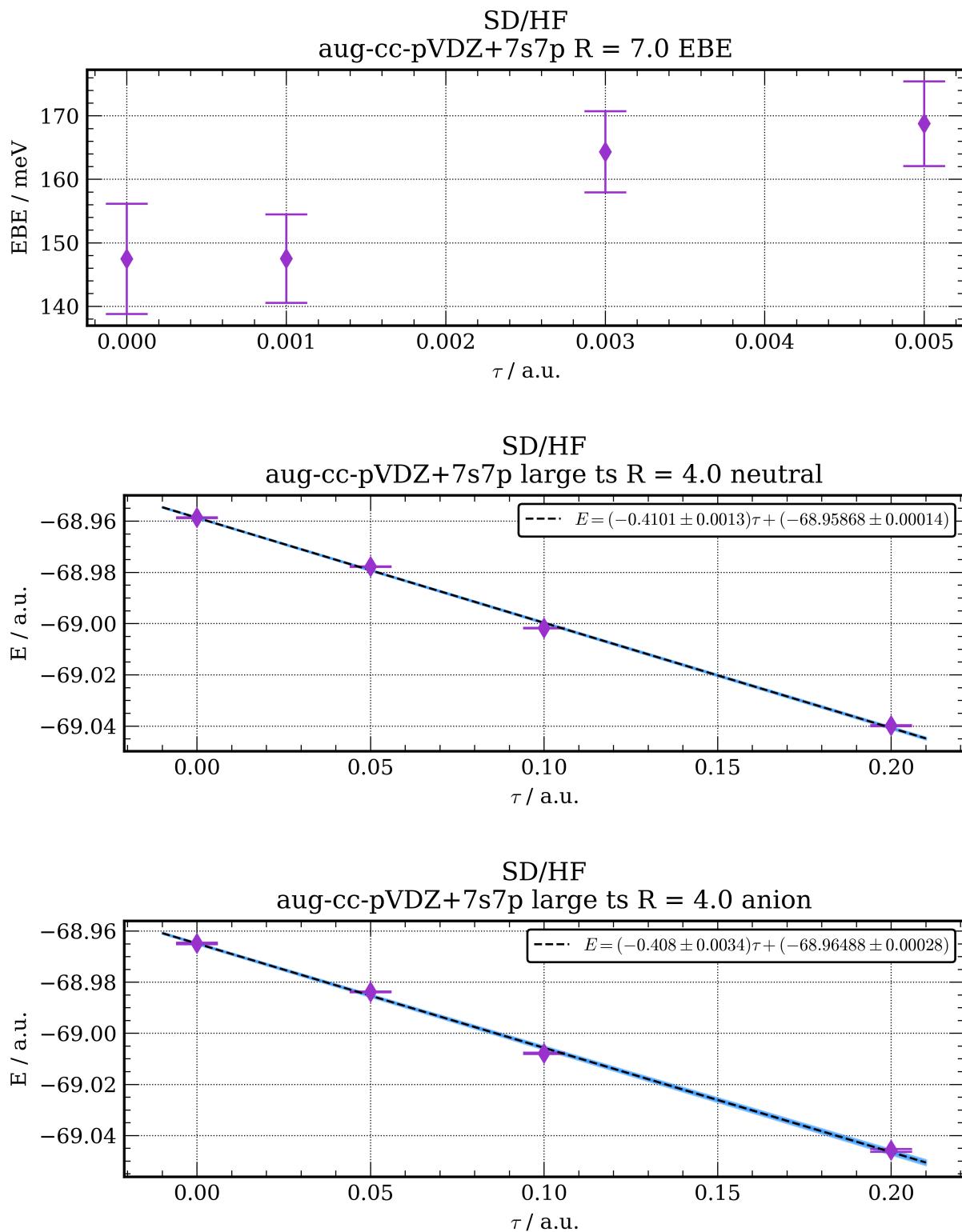
A summary of the fit for the zero time step linear extrapolation is plotted below. The blue shaded region corresponds to the error in the fit of the DMC energies at the three timesteps (0.001, 0.003, 0.005).

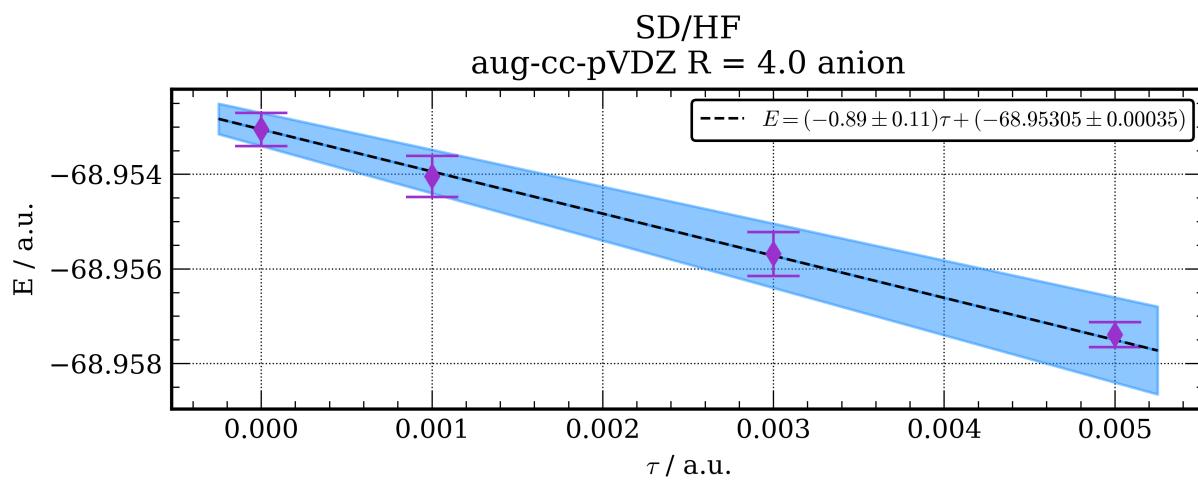
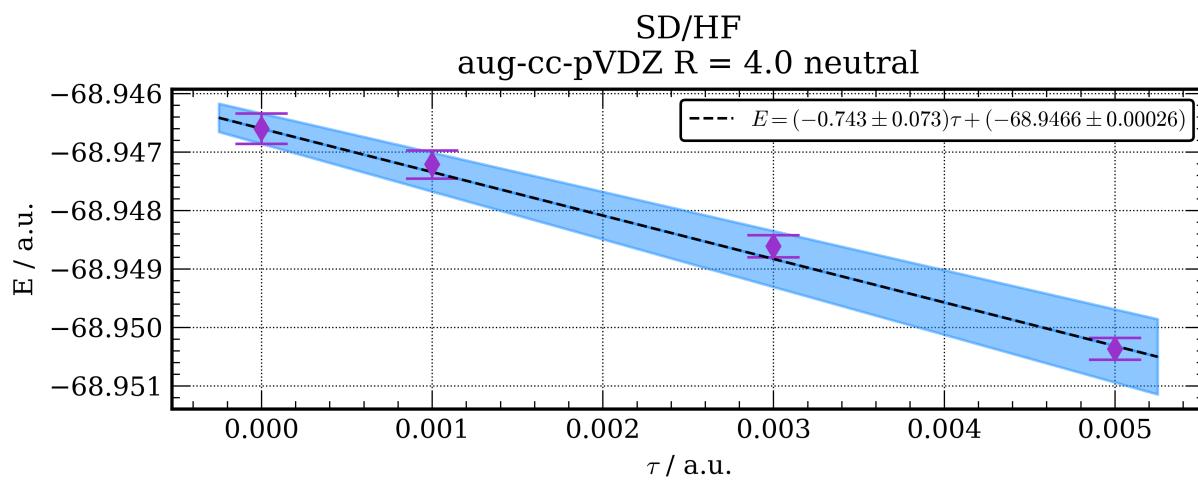
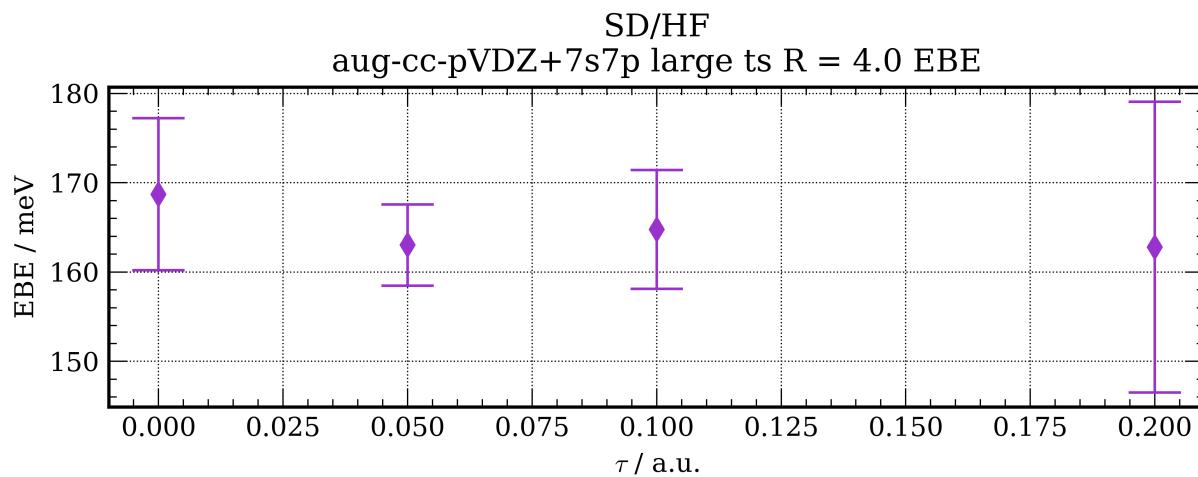


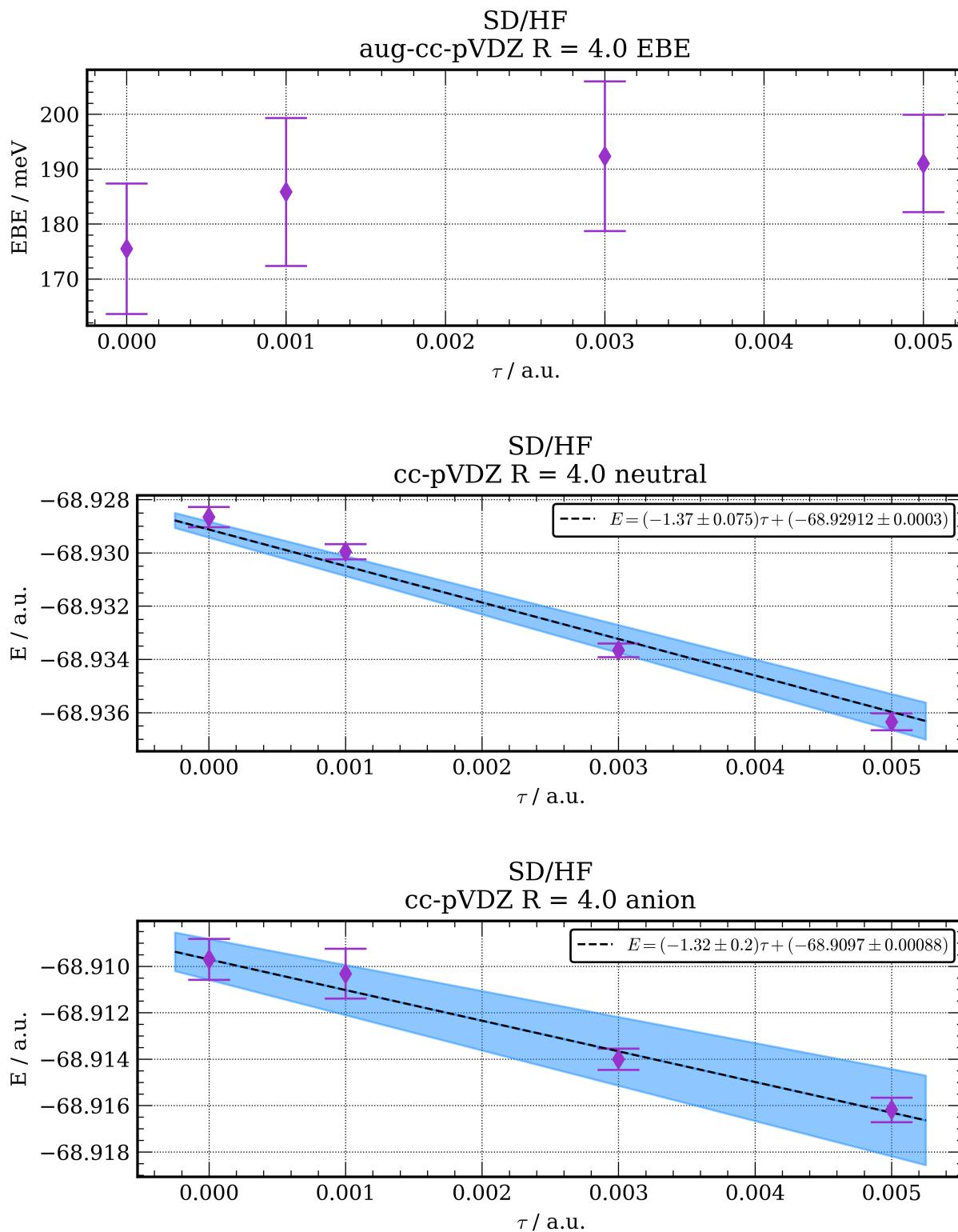


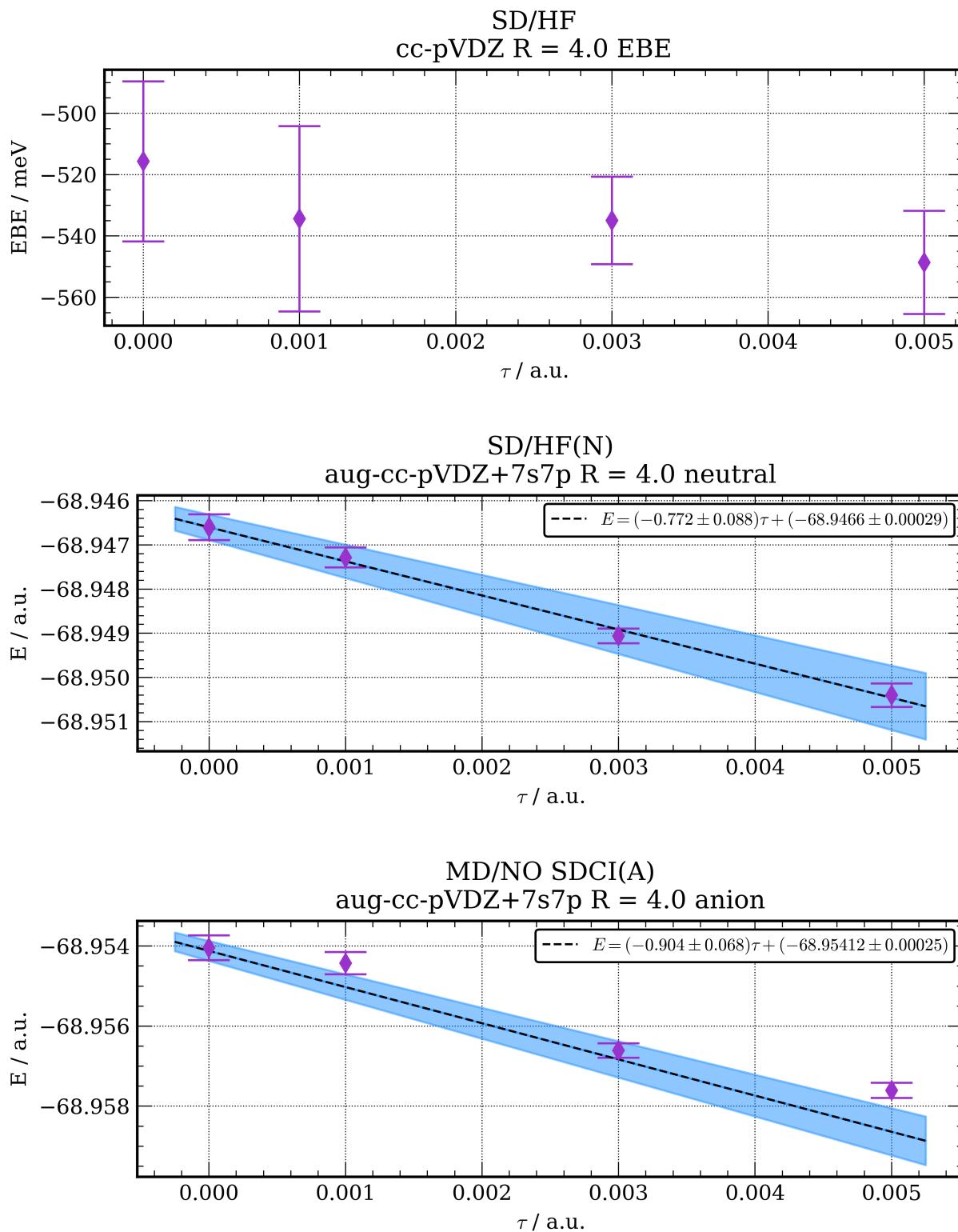


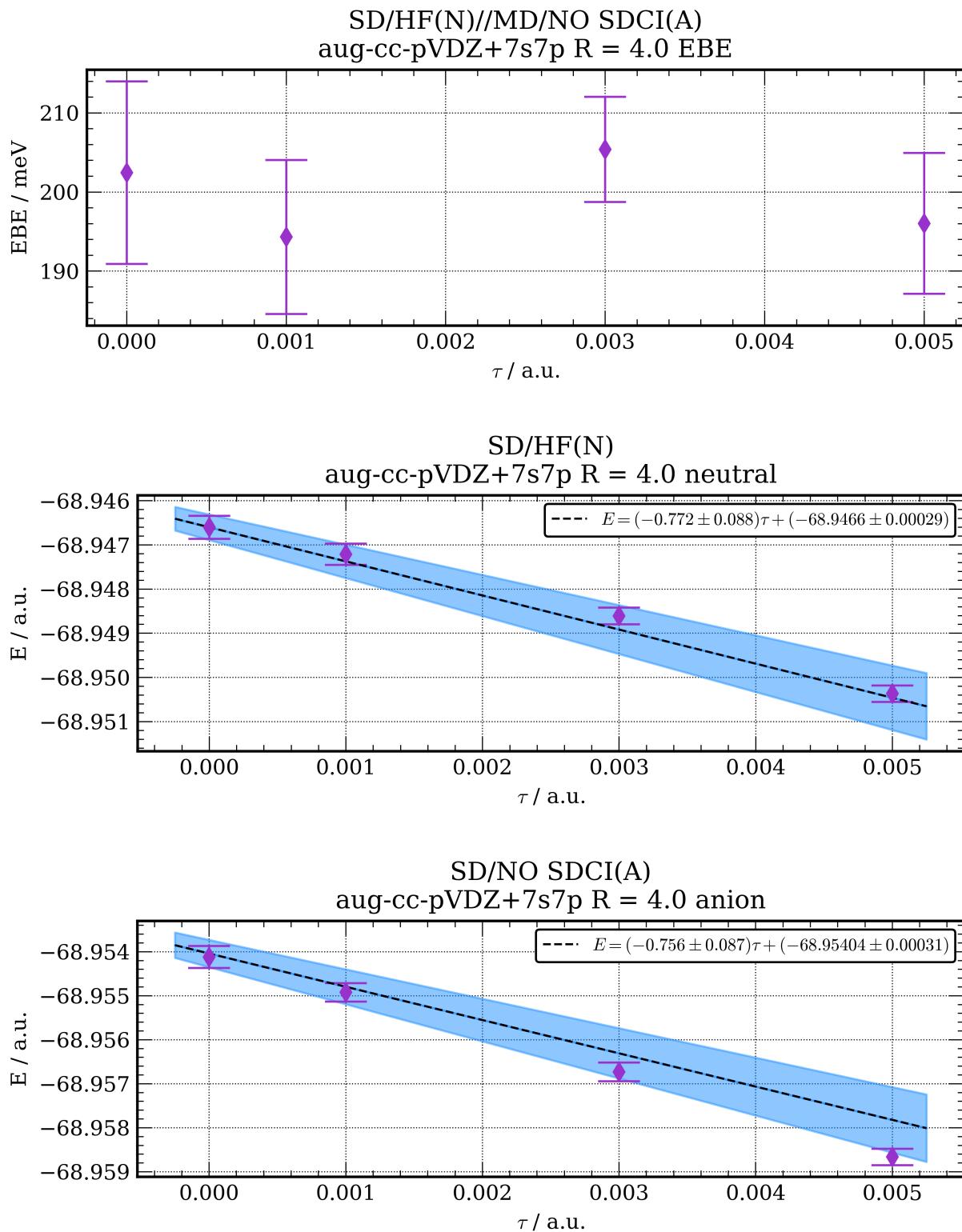


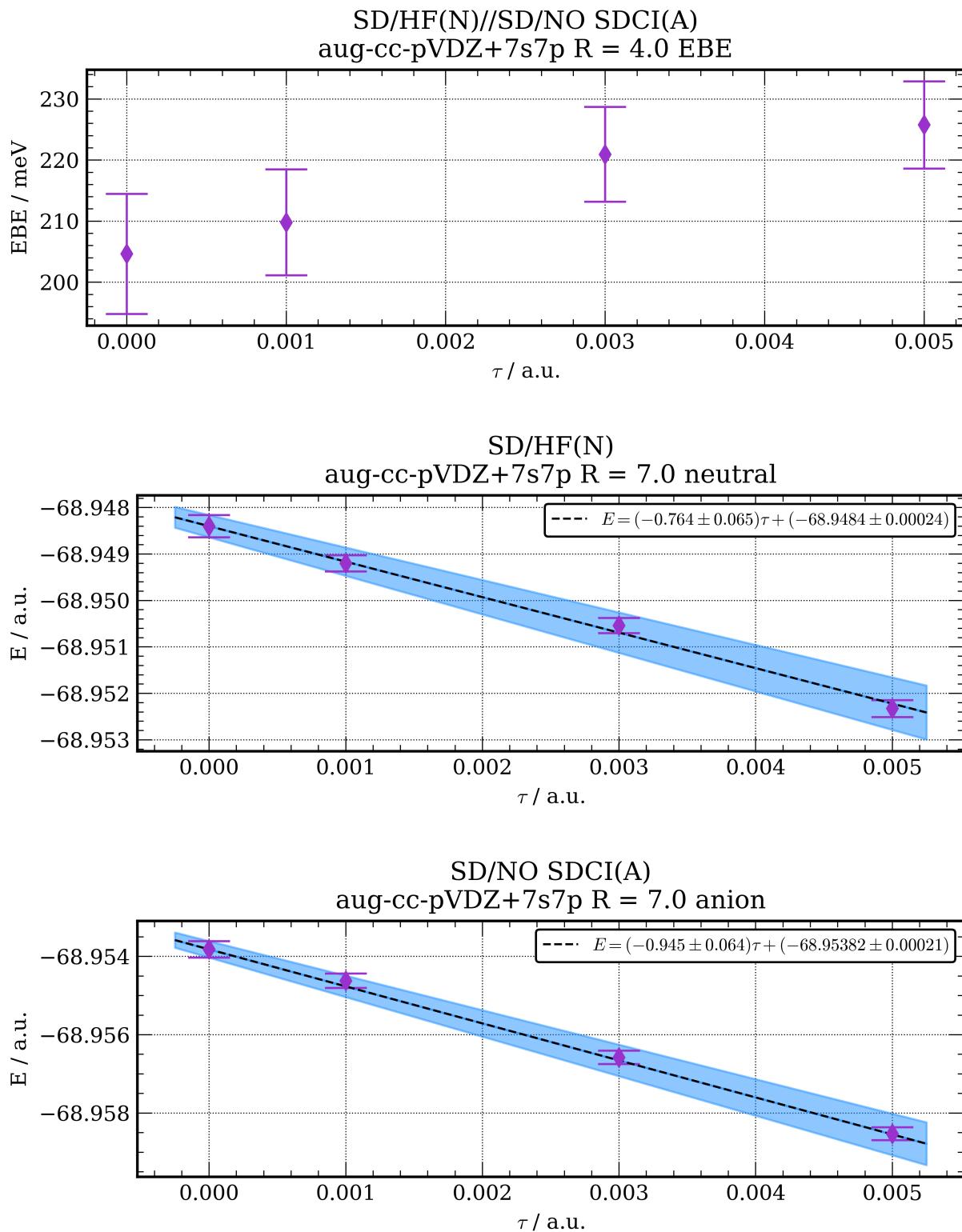


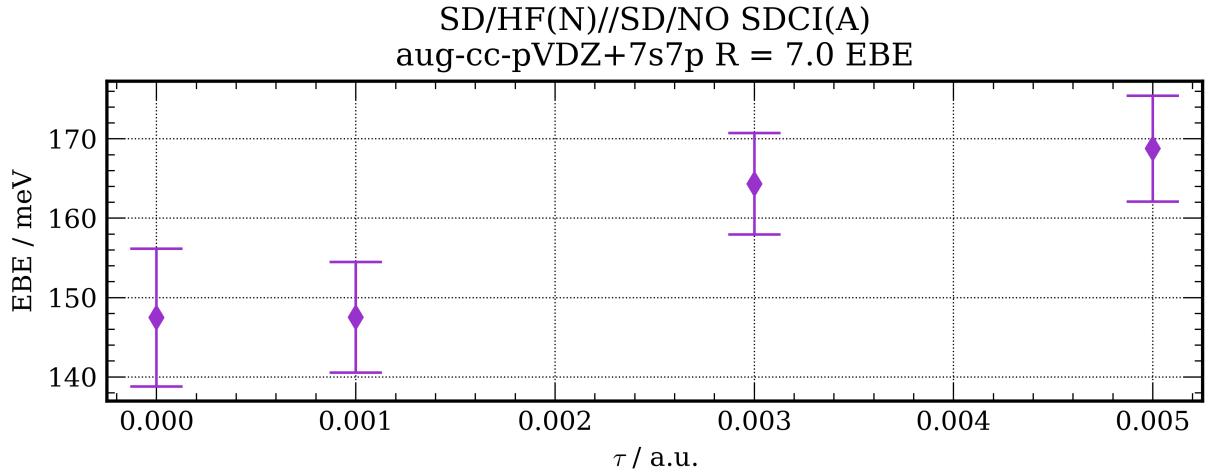












3 RADIAL ORBITAL DENSITY PLOTS

The radial orbital density, $\psi^2(r)$, plots are created by integrating over angular portion of the norm of the single particle wave function.

$$\psi^2(r) = \int_0^{2\pi} \int_0^\pi \psi^2(r, \theta, \phi) r^2 \sin(\theta) dr d\theta d\phi \quad (1)$$

Discretizing this expression using a uniform radial grid and a Lebedev-Laikov quadrature for the angular components, yields a form that can be readily evaluated.

$$\psi^2(r_i) = 4\pi r_i^2 \sum_j^{N^{ang}} w_j^{ang} \psi^2(r_i, \theta_j, \phi_j) \quad (2)$$

The function $\psi^2(r_i)$ from Equation 2 can be plotted with the points r_i serving as the abscissa. Since the singly occupied orbitals are normalized, the proximity of the sum of the radial quadrature to unity is used as a check.

$$\sum_i^{N^{rad}} \psi^2(r_i) w_i^{rad} = \sum_i^{N^{rad}} \psi^2(r_i) \Delta r \approx 1 \quad (3)$$

3.a Required software versions

| Required software | version |
|-------------------|---------|
| numpy | 1.18.4 |
| quadpy | 0.16.2 |
| pyscf | 1.7.0 |
| cclib | 1.6.3 |

3.b Step 1: Generating a Molden file

Molden files were generated using `cclib`, with the exception of the natural orbital from the CIPSI calculations. Since QuantumPackage is not supported by `cclib`, Molden files were created using the native utility in QuantumPackage 2.0. For the Molden files generated with `cclib`, the `-g/--ghost` flag indicates the presence of a ghost atom. By default the only molecular orbitals can be written to a Molden file, therefore the `-n/--naturalorbitals` flag was created to allow natural orbitals to be written in place of molecular orbitals. This flag is not yet available in the official distribution, but a request to incorporate it in the official distribution has been opened (<https://github.com/cclib/cclib/pull/948>).

```
$ ccwrite molden -g "X" -n QUANTUM_CHEMISTRY_OUTPUT_FILE
```

3.c Step 2: Integrating over the angular components of the singly occupied orbital

`quadpy` was used to generate the Lebedev-Laikov integration weights and points. The singly occupied molecular/natural orbital was evaluated at these points using PySCF.

```
import numpy  
import quadpy  
import pyscf  
import pyscf.tools  
  
filename = "FILENAME.molden"
```

```

r_max = 100

num_radial_pts = 1000

mo_idx = 20

# use pyscf to load the molden

mol, mo_energy, mo_coeff, mo_occ, irrep_labels, spins = pyscf.tools.molden.load(
    filename
)

# extract the singly occupied orbital coefficients

singly_occ_orb = mo_coeff[:, mo_idx]

# generate the angular points and weights using quadpy

lebedev_laikov = quadpy.u3.schemes["lebedev_131"]()

angular_pts = lebedev_laikov.theta_phi

angular_weights = lebedev_laikov.weights

num_angular_pts = len(angular_pts[0])

# generate the radial points and weights using numpy

radial_pts = numpy.linspace(r_max, 0, num_radial_pts, endpoint=False)[::-1]

radial_weights = numpy.ones_like(radial_pts) * (radial_pts[1] - radial_pts[0])

# a helper function to convert radial and angular points to cartesian

def sph2cart(r, theta_phi):

    theta = theta_phi[0]

    phi = theta_phi[1]

    x = r * numpy.cos(theta) * numpy.sin(phi)

    y = r * numpy.sin(theta) * numpy.sin(phi)

    z = r * numpy.cos(phi)

    return numpy.vstack((x, y, z)).T

# integrate over the angular points for each radial point

values = []

```

```
for r in radial_pts:
    r_pts = r * numpy.ones(num_angular_pts)
    coords = sph2cart(r_pts, angular_pts)
    ao = mol.eval_gto("GT0val_cart", coords)
    value = angular_weights @ ao @ singly_occ_orb
    values.append(4 * numpy.pi * r * r * value ** 2)

# output the values
values = numpy.array(values)

numpy.savetxt("{}_values.txt".format(filename), values)
numpy.savetxt("{}_r.txt".format(filename), radial_pts)

# check the norm of the orbital
print(radial_weights @ values)
```