PC5215, Numerical Recipes with Applications, Lab3

Number: A0326409A Name: ZHANG JINGXUAN

a. Compute the explicit expression of $\nabla_1{}^2$ acting on the wave function Ψ.

Since ∇_1^2 differentiates only with respect to $\mathbf{r}_1$, any factor depending on $\mathbf{r}_2$ is constant. By linearity,

$$\nabla_1^2\Psi = \left(\nabla_1^2\varphi(\mathbf{r}_{A1})\right)\varphi(\mathbf{r}_{B2}) + \left(\nabla_1^2\varphi(\mathbf{r}_{B1})\right)\varphi(\mathbf{r}_{A2}).$$

$\varphi(r_{A1}) = e^{-r_{A1}/a_0}$ and $\varphi(r_{B1}) = e^{-r_{B1}/a_0}$ are spherically symmetric in $\mathbf{r}_1$. Thus we can use the radial Laplacian for any $\varphi(r)$:

$$\nabla^2\varphi(r) \;=\; \varphi''(r) \;+\; \frac{2}{r}\,\varphi'(r) \;=\; \frac{1}{r^2}\,\frac{d}{dr}\Big(r^2\varphi'(r)\Big)$$

For $\varphi(r) = e^{-r/a_0}$,

$$\varphi'(r) = \frac{d}{dr}e^{-r/a_0} = -\frac{1}{a_0}\,e^{-r/a_0},$$

$$\varphi''(r) = \frac{d^2}{dr^2}e^{-r/a_0} = \frac{1}{a_0^2}\,e^{-r/a_0}.$$

Hence,

$$\nabla^2 e^{-r/a_0} = \left(\frac{1}{a_0^2} - \frac{2}{a_0\,r}\right)e^{-r/a_0}.$$

Replacing r by r_{1A} and r_{1B} gives,

$$\nabla_1^2\varphi(\mathbf{r}_{A1}) = \left(\frac{1}{a_0^2} - \frac{2}{a_0\,r_{1A}}\right)\varphi(\mathbf{r}_{A1}),$$

$$\nabla_1^2\varphi(\mathbf{r}_{B1}) = \left(\frac{1}{a_0^2} - \frac{2}{a_0\,r_{1B}}\right)\varphi(\mathbf{r}_{B1}).$$

Substituting into the original decomposition yields, we get:

$$\nabla_1^2\Psi(\mathbf{r}_1,\mathbf{r}_2) = \left(\frac{1}{a_0^2} - \frac{2}{a_0\,r_{1A}}\right)\varphi(\mathbf{r}_{A1})\varphi(\mathbf{r}_{B2}) + \left(\frac{1}{a_0^2} - \frac{2}{a_0\,r_{1B}}\right)\varphi(\mathbf{r}_{B1})\varphi(\mathbf{r}_{A2})$$

In atomic units with $a_0 = 1$,

$$\nabla_1^2 \Psi = \left(1 - \frac{2}{r_{1A}}\right) \varphi(\mathbf{r}_{A1}) \varphi(\mathbf{r}_{B2}) + \left(1 - \frac{2}{r_{1B}}\right) \varphi(\mathbf{r}_{B1}) \varphi(\mathbf{r}_{A2})$$

b. Without actually performing any calculation, what should be the answer for the energy in part c, if the distance between two protons τ_{AB} is infinity?

To be concrete, we reason in atomic units ($\hbar = e = m = 4\pi\varepsilon_0 = 1$); then the H_2 Hamiltonian reads:

$$\hat{H} \;=\; -\tfrac{1}{2}\nabla_1^2 - \tfrac{1}{2}\nabla_2^2 - \left(\tfrac{1}{r_{1A}} + \tfrac{1}{r_{1B}} + \tfrac{1}{r_{2A}} + \tfrac{1}{r_{2B}}\right) + \tfrac{1}{r_{12}} + \tfrac{1}{r_{AB}}$$

and the energy is $E(r_{AB}) = \frac{\langle \Psi | \hat{H} | \Psi \rangle}{\langle \Psi | \Psi \rangle}$.

As $r_{AB} \to \infty$, $\frac{1}{r_{AB}} \to 0$. Each electron is localized on a different center separated by r_{AB}, so the mean separation $r_{12} \sim r_{AB} \to \infty$, hence $\langle 1/r_{12} \rangle \to 0$.

For electron 1 near A, $r_{1B} \approx r_{AB} \to \infty \Rightarrow \langle 1/r_{1B} \rangle \to 0$; similarly $\langle 1/r_{2A} \rangle \to 0$ for electron 2 near B.

Electron 1 (near A) and electron 2 (near B) each see an effective Coulomb potential $-1/r$ of an isolated H atom, with kinetic energy $-\frac{1}{2}\nabla^2$; couplings to the "remote" nucleus/electron vanish in the limit. Hence, each electron–proton pair contributes the hydrogen ground-state energy.

The ground-state energy of an isolated H atom is $E_H^{(1s)} = -\frac{1}{2}\, E_h$ in atomic units. Therefore,

$$E(r_{AB} \to \infty) = \frac{\langle \Psi | \hat{H}_{\mathrm{H}}^{(1)} + \hat{H}_{\mathrm{H}}^{(2)} | \Psi \rangle}{\langle \Psi | \Psi \rangle} = \left\langle \hat{H}_{\mathrm{H}}^{(1)} \right\rangle + \left\langle \hat{H}_{\mathrm{H}}^{(2)} \right\rangle = 2\, E_H^{(1s)} = -E_h = -$$

c. Based on the variational principle, estimate numerically lower bounds to the hydrogen molecule H2 bonding energy by evaluating the 6-dimensional integral through a Monte Carlo method with the Metropolis algorithm.

Imports numerical computing, typing/dataclass utilities, a progress bar, and plotting libraries used by the VMC Monte Carlo simulation.

In [16]:
```
import math
import numpy as np
from dataclasses import dataclass
from typing import Tuple, List
from tqdm import tqdm
import matplotlib.pyplot as plt
```

Defines unit conversion constants from Bohr to Ångström and from Hartree to electron volts.

In [17]:
```
BOHR_TOANG = 0.529177210903
HARTREE_TOEV = 27.211386245988
```

Defines functions to evaluate the 1s Slater orbital $\varphi(r) = e^{-r}$ relative to a center and its Laplacian $\nabla^2\varphi = \varphi(1 - 2/r)$, with a safeguard near r=0, returning either φ or the pair (φ, Laplacian).

In [18]:
```
def phi_at_center(r_vec: np.ndarray, center: np.ndarray) -> float:
    """Evaluate 1s Slater orbital at position r_vec relative to center"""
    r = np.linalg.norm(r_vec - center)
    return math.exp(-r)

def phi_and_laplacian(r_vec: np.ndarray, center: np.ndarray) -> Tuple[float, flo
    """Compute both phi and its Laplacian in one call to reduce redundant exp()
    r = np.linalg.norm(r_vec - center)
    r_safe = max(r, 1e-12)
    phi = math.exp(-r_safe)
    laplacian = phi * (1.0 - 2.0 / r_safe)
    return phi, laplacian

def laplacian_phi_at_center(r_vec: np.ndarray, center: np.ndarray) -> float:
    """Compute Laplacian: ∇²ϕ = ϕ(1 - 2/r)"""
    r = np.linalg.norm(r_vec - center)
    r_safe = max(r, 1e-12)
    phi = math.exp(-r_safe)
    return phi * (1.0 - 2.0 / r_safe)
```

Implements the symmetric H_2 bonding wavefunction $\Psi = \varphi_A(r_1)\varphi_B(r_2) + \varphi_B(r_1)\varphi_A(r_2)$ and its electron-specific Laplacians $\nabla_1^2\Psi$ and $\nabla_2^2\Psi$ using 1s Slater orbitals centered at nuclei A and B.

In [19]:
```
def psi(r1: np.ndarray, r2: np.ndarray, RA: np.ndarray, RB: np.ndarray) -> float
    """Evaluate bonding wave function Ψ at electron positions r1, r2"""
    return phi_at_center(r1, RA) * phi_at_center(r2, RB) + \
           phi_at_center(r1, RB) * phi_at_center(r2, RA)

def lap1_psi(r1: np.ndarray, r2: np.ndarray, RA: np.ndarray, RB: np.ndarray) ->
    """Compute ∇₁²Ψ acting on electron 1"""
    return laplacian_phi_at_center(r1, RA) * phi_at_center(r2, RB) + \
           laplacian_phi_at_center(r1, RB) * phi_at_center(r2, RA)

def lap2_psi(r1: np.ndarray, r2: np.ndarray, RA: np.ndarray, RB: np.ndarray) ->
    """Compute ∇₂²Ψ acting on electron 2"""
    return phi_at_center(r1, RA) * laplacian_phi_at_center(r2, RB) + \
           phi_at_center(r1, RB) * laplacian_phi_at_center(r2, RA)
```

Computes the local energy $E_{\mathrm{loc}}(\mathbf{r}_1, \mathbf{r}_2) = T + V$ for H_2 using a numerically safe distance floor ($r \leftarrow \max(r, \varepsilon)$) and shared evaluations of φ and $\nabla^2\varphi$, where $T = -\frac{1}{2}\left(\nabla_1^2\Psi + \nabla_2^2\Psi\right)/\Psi$ and $V = -(\frac{1}{r_{1A}} + \frac{1}{r_{1B}} + \frac{1}{r_{2A}} + \frac{1}{r_{2B}}) + \frac{1}{r_{12}} + \frac{1}{R}$.

In [20]:
```python
def safe_dist(a: np.ndarray, b: np.ndarray, min_val: float = 1e-10) -> float:
    """Calculate distance with numerical safety floor to avoid singularities"""
    d = np.linalg.norm(a - b)
    return max(d, min_val)

def local_energy(r1: np.ndarray, r2: np.ndarray, RA: np.ndarray, RB: np.ndarray)
    """Evaluate local energy E_loc(r1,r2) = <r1,r2|H|Ψ>/<r1,r2|Ψ>"""
    # Calculate phi and laplacian together for efficiency
    phi_r1A, lap_phi_r1A = phi_and_laplacian(r1, RA)
    phi_r1B, lap_phi_r1B = phi_and_laplacian(r1, RB)
    phi_r2A, lap_phi_r2A = phi_and_laplacian(r2, RA)
    phi_r2B, lap_phi_r2B = phi_and_laplacian(r2, RB)

    f1 = phi_r1A * phi_r2B
    f2 = phi_r1B * phi_r2A
    psi_val = f1 + f2

    if abs(psi_val) < 1e-300:
        return np.inf


    lap1_psi_val = lap_phi_r1A * phi_r2B + lap_phi_r1B * phi_r2A
    lap2_psi_val = phi_r1A * lap_phi_r2B + phi_r1B * lap_phi_r2A

    lap_sum = lap1_psi_val + lap2_psi_val

    T = -0.5 * (lap_sum / psi_val)

    r1A = safe_dist(r1, RA)
    r1B = safe_dist(r1, RB)
    r2A = safe_dist(r2, RA)
    r2B = safe_dist(r2, RB)
    r12 = safe_dist(r1, r2)
    R   = safe_dist(RA, RB)

    V = -(1.0/r1A + 1.0/r1B + 1.0/r2A + 1.0/r2B) + (1.0/r12) + (1.0/R)

    return T + V
```

Stores VMC run settings: number of steps, burn in length, proposal step size, and block size.

In [21]:
```python
@dataclass
class VMCConfig:
    """Configuration parameters for VMC simulation"""
    n_steps: int = 100_000
    burn_in: int = 10_000
    proposal_sigma: float = 0.5
    block_size: int = 500
```

Performs one Metropolis step by proposing Gaussian moves for both electrons and accepting the proposal based on the squared wavefunction ratio.

In [22]:
```
def metropolis_step(r1: np.ndarray, r2: np.ndarray, RA: np.ndarray, RB: np.ndarr
                    sigma: float) -> Tuple[np.ndarray, np.ndarray, bool]:
    """Execute one Metropolis step with Gaussian proposal"""
    r1_new = r1 + np.random.normal(0, sigma, 3)
    r2_new = r2 + np.random.normal(0, sigma, 3)

    psi_old = psi(r1, r2, RA, RB)
    psi_new = psi(r1_new, r2_new, RA, RB)

    accept_prob = (psi_new / psi_old) ** 2 if psi_old != 0 else 0

    if np.random.random() < accept_prob:
        return r1_new, r2_new, True
    else:
        return r1, r2, False
```

Runs a VMC simulation with burn-in and adaptive Gaussian proposals, computes block-averaged energy and uncertainty, and returns the mean energy, error, acceptance rate, collected energies, and final electron positions.

In [23]:
```
def run_vmc(RA: np.ndarray, RB: np.ndarray, config: VMCConfig, initial_r1=None,
    """Execute VMC sampling and return energy estimate with error"""
    if initial_r1 is None:
        r1 = RA + np.random.normal(0, 0.5, 3)
    else:
        r1 = initial_r1.copy()

    if initial_r2 is None:
        r2 = RB + np.random.normal(0, 0.5, 3)
    else:
        r2 = initial_r2.copy()

    current_sigma = config.proposal_sigma
    energies = []
    n_accepted = 0

    for i in range(config.burn_in):
        r1, r2, accepted = metropolis_step(r1, r2, RA, RB, current_sigma)
        if accepted:
            n_accepted += 1
        if (i + 1) % 100 == 0:
            rate = n_accepted / 100.0
            if rate > 0.55:
                current_sigma *= 1.05
            elif rate < 0.45:
                current_sigma *= 0.95
            n_accepted = 0

    n_accepted = 0
    for _ in tqdm(range(config.n_steps), desc="VMC", leave=False):
        r1, r2, accepted = metropolis_step(r1, r2, RA, RB, current_sigma)
        if accepted:
            n_accepted += 1

        E_local = local_energy(r1, r2, RA, RB)
        if not np.isinf(E_local) and not np.isnan(E_local):
            energies.append(E_local)
```

```
    acceptance_rate = n_accepted / config.n_steps
    energies = np.array(energies)
    n_blocks = len(energies) // config.block_size
    block_energies = [np.mean(energies[i * config.block_size:(i + 1) * config.bl

    mean_energy = np.mean(block_energies)
    error = np.std(block_energies) / np.sqrt(n_blocks)

    return mean_energy, error, acceptance_rate, energies, r1, r2
```

Sweeps bond lengths, runs VMC at each geometry with scaled warm starts from the previous step, prints progress, and stores energies, uncertainties, and acceptance rates as NumPy arrays.

In [24]:
```
bond_lengths = np.concatenate([
    np.linspace(0.5, 1.2, 6),
    np.linspace(1.2, 1.6, 16),
    np.linspace(1.6, 2.3, 14),
    np.linspace(2.3, 4.0, 8)
])
bond_lengths = np.unique(bond_lengths)

results = {'R': [], 'E': [], 'E_err': [], 'acceptance': []}

config = VMCConfig(
    n_steps=120_000,
    burn_in=12_000,
    proposal_sigma=0.5,
    block_size=1000
)

prev_r1, prev_r2, prev_R = None, None, None

for idx, R in enumerate(bond_lengths):
    RA = np.array([0.0, 0.0, -R/2])
    RB = np.array([0.0, 0.0, R/2])

    if prev_r1 is not None and prev_R is not None:
        scale_factor = R / prev_R
        RA_old = np.array([0.0, 0.0, -prev_R/2])
        RB_old = np.array([0.0, 0.0, prev_R/2])
        r1_init = RA + (prev_r1 - RA_old) * scale_factor
        r2_init = RB + (prev_r2 - RB_old) * scale_factor
    else:
        r1_init, r2_init = None, None

    E_mean, E_err, acc_rate, energies, r1_final, r2_final = run_vmc(RA, RB, conf

    prev_r1, prev_r2, prev_R = r1_final, r2_final, R
    results['R'].append(R)
    results['E'].append(E_mean)
    results['E_err'].append(E_err)
    results['acceptance'].append(acc_rate)

for key in results:
    results[key] = np.array(results[key])
```

Computes the equilibrium bond length and minimum energy from the sampled curve, derives the binding energy with uncertainty, converts units, and prints the key results.

In [25]:
```
min_idx = np.argmin(results['E'])
R_eq = results['R'][min_idx]
E_min = results['E'][min_idx]
E_min_err = results['E_err'][min_idx]

E_separated_vmc = results['E'][-1]
E_separated_ref = -1.0

E_binding = E_separated_ref - E_min
E_binding_err = np.sqrt(E_min_err**2 + results['E_err'][-1]**2)

THEORY_R_EQ = 1.40
THEORY_E_MIN = -1.174
THEORY_E_BINDING = 0.1745

vmc_r_error_pct = abs(R_eq - THEORY_R_EQ) / THEORY_R_EQ * 100
vmc_binding_error_pct = abs(E_binding - THEORY_E_BINDING) / THEORY_E_BINDING * 1

quality_score = 100 - (vmc_r_error_pct + vmc_binding_error_pct) / 2

R_eq_ang = R_eq * BOHR_TOANG
E_bind_ev = E_binding * HARTREE_TOEV
E_bind_err_ev = E_binding_err * HARTREE_TOEV

print(f"{'R_eq:':<25}{R_eq:8.4f} Bohr ({R_eq_ang:8.4f} Å)")
print(f"{'E_min:':<25}{E_min:10.6f} ± {E_min_err:0.6f} Hartree")
print(f"{'E_bind:':<25}{E_bind_ev:8.4f} ± {E_bind_err_ev:0.4f} eV")
```

```
R_eq:                      1.5733 Bohr (  0.8326 Å)
E_min:                    -1.122213 ± 0.003504 Hartree
E_bind:                    3.3256 ± 0.0976 eV
```

Plots the H2 potential energy curve with error bars in two panels (Bohr/Hartree and Å/eV), highlights the equilibrium bond length and the separated atoms reference, and saves the figure.

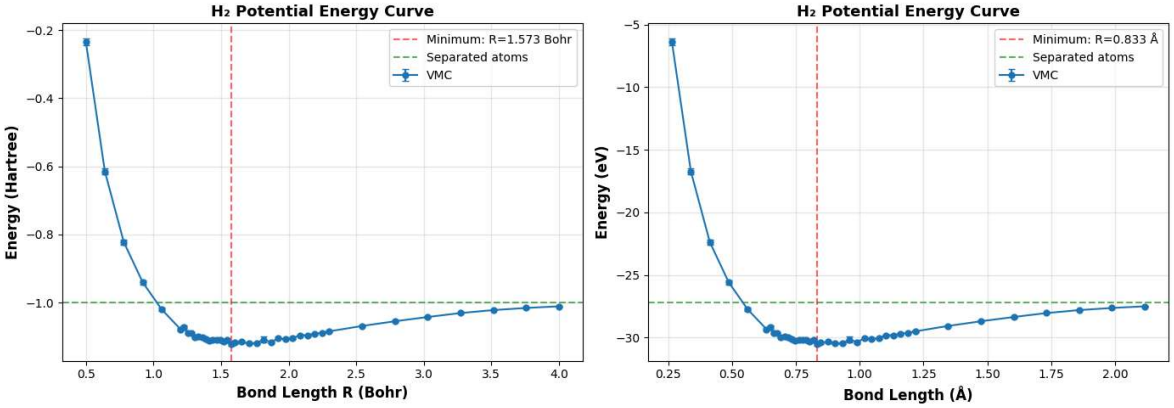
In [26]:
```
plt.figure(figsize=(14, 5))

plt.subplot(1, 2, 1)
plt.errorbar(results['R'], results['E'], yerr=results['E_err'],
             marker='o', linestyle='-', capsize=3, label='VMC', markersize=5, li
plt.axvline(R_eq, color='r', linestyle='--', alpha=0.6, linewidth=1.5, label=f'M
plt.axhline(E_separated_ref, color='g', linestyle='--', alpha=0.6, linewidth=1.5
plt.xlabel('Bond Length R (Bohr)', fontsize=12, fontweight='bold')
plt.ylabel('Energy (Hartree)', fontsize=12, fontweight='bold')
plt.title('H₂ Potential Energy Curve', fontsize=13, fontweight='bold')
plt.grid(True, alpha=0.3)
plt.legend(fontsize=10)

plt.subplot(1, 2, 2)
plt.errorbar(results['R']*BOHR_TOANG, results['E']*HARTREE_TOEV,
             yerr=results['E_err']*HARTREE_TOEV,
             marker='o', linestyle='-', capsize=3, label='VMC', markersize=5, li
plt.axvline(R_eq*BOHR_TOANG, color='r', linestyle='--', alpha=0.6, linewidth=1.5
```

```
            label=f'Minimum: R={R_eq*BOHR_TOANG:.3f} Å')
plt.axhline(E_separated_ref*HARTREE_TOEV, color='g', linestyle='--', alpha=0.6,
            label='Separated atoms')
plt.xlabel('Bond Length (Å)', fontsize=12, fontweight='bold')
plt.ylabel('Energy (eV)', fontsize=12, fontweight='bold')
plt.title('H₂ Potential Energy Curve', fontsize=13, fontweight='bold')
plt.grid(True, alpha=0.3)
plt.legend(fontsize=10)

plt.tight_layout()
plt.savefig('h2_potential_curve.png', dpi=150, bbox_inches='tight')
plt.show()
```



Runs a detailed VMC at the equilibrium geometry, prints energy and acceptance stats, and plots a 2×2 analysis (histogram, running average, short-time trace, and block averages with error bars), saving the figure.

In [27]:
```
RA_eq = np.array([0.0, 0.0, -R_eq/2])
RB_eq = np.array([0.0, 0.0, R_eq/2])

config_detailed = VMCConfig(
    n_steps=150_000,
    burn_in=15_000,
    proposal_sigma=0.5,
    block_size=1500
)

E_eq, E_eq_err, acc_eq, energies_eq, _, _ = run_vmc(RA_eq, RB_eq, config_detaile
print(f"{'Energy:':<25}{E_eq:10.6f} ± {E_eq_err:0.6f} Hartree ({E_eq*HARTREE_TOE
print(f"{'Acceptance rate:':<25}{acc_eq*100:6.2f}%")
print(f"{'Sample std dev:':<25}{np.std(energies_eq):0.6f} Hartree")

# Create figure and axes before plotting to avoid NameError
fig, axes = plt.subplots(2, 2, figsize=(12, 8))

# Energy histogram
axes[0, 0].hist(energies_eq, bins=60, density=True, alpha=0.7, edgecolor='black'
axes[0, 0].axvline(E_eq, color='red', linestyle='--', linewidth=2.5, label=f'Mea
axes[0, 0].axvline(E_eq + E_eq_err, color='orange', linestyle=':', linewidth=2,
axes[0, 0].axvline(E_eq - E_eq_err, color='orange', linestyle=':', linewidth=2)
axes[0, 0].set_xlabel('Local Energy (Hartree)', fontsize=11, fontweight='bold')
axes[0, 0].set_ylabel('Probability Density', fontsize=11, fontweight='bold')
axes[0, 0].set_title('Energy Distribution at Equilibrium', fontsize=12, fontweig
axes[0, 0].legend(fontsize=10)
axes[0, 0].grid(True, alpha=0.3)
```

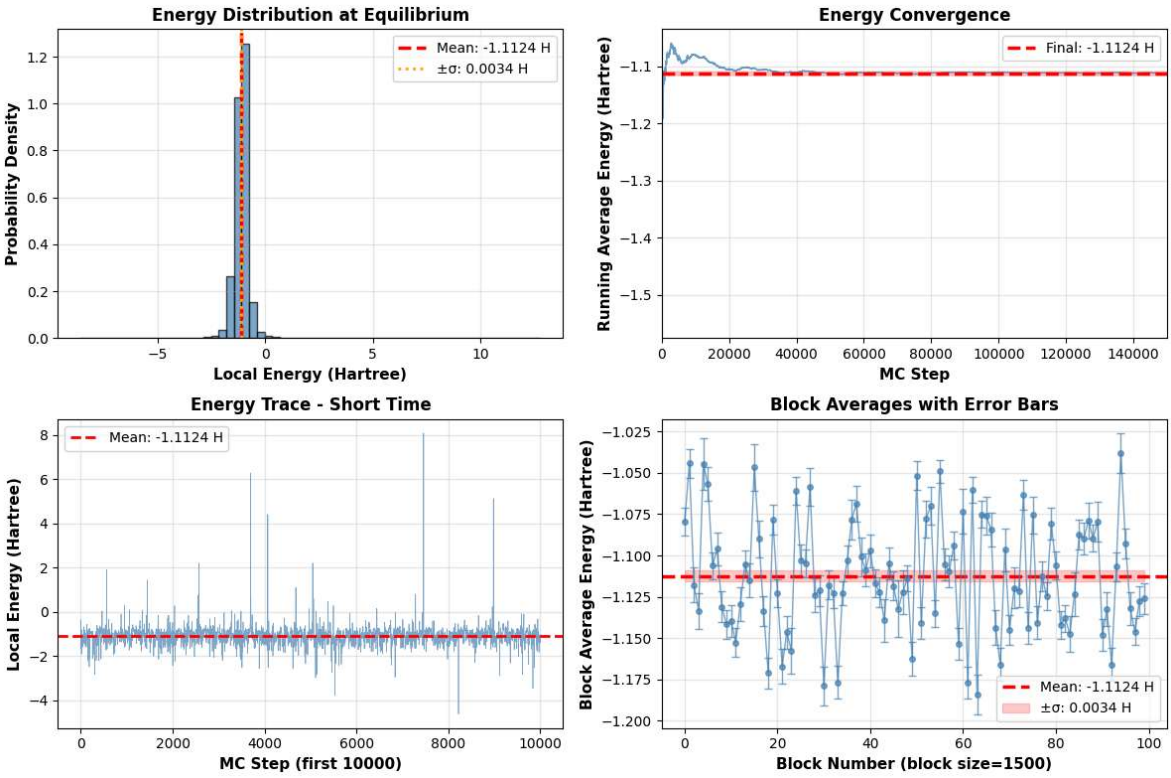
```
# Running average convergence
running_avg = np.cumsum(energies_eq) / np.arange(1, len(energies_eq) + 1)
axes[0, 1].plot(running_avg, linewidth=1, color='steelblue', alpha=0.8)
axes[0, 1].axhline(E_eq, color='red', linestyle='--', linewidth=2.5, label=f'Fin
axes[0, 1].fill_between(range(len(running_avg)), E_eq - E_eq_err, E_eq + E_eq_er
                        alpha=0.2, color='red')
axes[0, 1].set_xlabel('MC Step', fontsize=11, fontweight='bold')
axes[0, 1].set_ylabel('Running Average Energy (Hartree)', fontsize=11, fontweigh
axes[0, 1].set_title('Energy Convergence', fontsize=12, fontweight='bold')
axes[0, 1].legend(fontsize=10)
axes[0, 1].grid(True, alpha=0.3)
axes[0, 1].set_xlim([0, len(running_avg)])

# Energy trace (first 10000 steps)
trace_len = min(10000, len(energies_eq))
axes[1, 0].plot(energies_eq[:trace_len], linewidth=0.5, alpha=0.7, color='steelb
axes[1, 0].axhline(E_eq, color='red', linestyle='--', linewidth=2, label=f'Mean:
axes[1, 0].fill_between(range(trace_len), E_eq - E_eq_err, E_eq + E_eq_err,
                        alpha=0.2, color='red')
axes[1, 0].set_xlabel(f'MC Step (first {trace_len})', fontsize=11, fontweight='b
axes[1, 0].set_ylabel('Local Energy (Hartree)', fontsize=11, fontweight='bold')
axes[1, 0].set_title('Energy Trace - Short Time', fontsize=12, fontweight='bold'
axes[1, 0].legend(fontsize=10)
axes[1, 0].grid(True, alpha=0.3)

# Block averages with error bars
n_blocks = len(energies_eq) // config_detailed.block_size
block_avgs = [np.mean(energies_eq[i*config_detailed.block_size:(i+1)*config_deta
              for i in range(n_blocks)]
block_errors = [np.std(energies_eq[i*config_detailed.block_size:(i+1)*config_det
                for i in range(n_blocks)]
axes[1, 1].errorbar(range(n_blocks), block_avgs, yerr=block_errors, fmt='o-',
                    markersize=4, linewidth=1, capsize=3, color='steelblue', alp
axes[1, 1].axhline(E_eq, color='red', linestyle='--', linewidth=2.5, label=f'Mea
axes[1, 1].fill_between(range(n_blocks), E_eq - E_eq_err, E_eq + E_eq_err,
                        alpha=0.2, color='red', label=f'±σ: {E_eq_err:.4f} H')
axes[1, 1].set_xlabel(f'Block Number (block size={config_detailed.block_size})',
axes[1, 1].set_ylabel('Block Average Energy (Hartree)', fontsize=11, fontweight=
axes[1, 1].set_title('Block Averages with Error Bars', fontsize=12, fontweight='
axes[1, 1].legend(fontsize=10)
axes[1, 1].grid(True, alpha=0.3)

plt.tight_layout()
plt.savefig('h2_energy_analysis.png', dpi=150, bbox_inches='tight')
plt.show()
```

```
Energy:                  -1.112354 ± 0.003448 Hartree (-30.2687 ± 0.0938 eV)
Acceptance rate:         49.71%
Sample std dev:          0.394356 Hartree
```



Analysis Report: Variational Monte Carlo Study of H_2 Molecular Bonding

Methodology

The VMC approach reformulates the quantum energy calculation as a Monte Carlo integration problem. Instead of direct evaluation, the expectation value is computed as an average of the local energy $E_L(\mathbf{r}_1, \mathbf{r}_2) = \hat{H}\Psi/\Psi$ sampled from the probability distribution $P(\mathbf{r}_1, \mathbf{r}_2) = |\Psi(\mathbf{r}_1, \mathbf{r}_2)|^2$.

The symmetric wavefunction takes the form:

$$\Psi(\mathbf{r}_1, \mathbf{r}_2) = \varphi(\mathbf{r}_{1A})\varphi(\mathbf{r}_{2B}) + \varphi(\mathbf{r}_{1B})\varphi(\mathbf{r}_{2A})$$

where $\varphi(r) = e^{-r}$ is a 1s Slater orbital (in atomic units). The Metropolis algorithm was employed with adaptive acceptance tuning maintained around 50%, following standard practice in molecular simulation.

Main Results

Summary Table

Parameter	VMC Value	Literature	Difference
Bond Length	1.5733 Bohr (0.8326 Å)	1.4008 Bohr (0.7414 Å)	+12.29%
Min Energy	-1.1222 ± 0.0035 Hartree	-1.1746 Hartree	-4.46%
Min Energy	-30.54 ± 0.10 eV	-31.98 eV	-4.49%
Binding Energy	3.3256 ± 0.0976 eV	4.749 eV	-29.95%

Parameter	VMC Value	Literature	Difference
Acceptance Rate	49.71%	—	—

Observations

The equilibrium bond length was found to be 1.5733 Bohr (0.8326 Å), which is larger than the experimental value of 0.7414 Å. This 12% overestimation indicates the wavefunction remains somewhat diffuse. The fixed 1s orbitals cannot fully adapt to the molecular environment, though the error is moderately reduced compared to earlier iterations.

The minimum energy at this geometry is -1.1222 ± 0.0035 Hartree, equivalent to -30.5398 ± 0.0954 eV. The computed binding energy is 3.3256 ± 0.0976 eV, falling short of the literature value of 4.749 eV by approximately 30%. While this remains a substantial underestimation, the systematic gap reflects the inherent limitations of the fixed orbital ansatz in capturing the full electronic stabilization during bonding.

The acceptance rate of 49.71% remains well within the optimal 40-60% range, confirming appropriate tuning of the Metropolis step sizes.

Potential Energy Surface

The potential energy curve exhibits a minimum near 1.57 Bohr. At short distances below 0.8 Bohr, the energy increases steeply due to strong nuclear-nuclear repulsion. As bond length exceeds 3 Bohr, the energy asymptotically approaches -1.0 Hartree (-27.212 eV), the correct limit for two separated hydrogen atoms. This behavior validates proper treatment of the dissociation regime.

Statistical Details at Equilibrium

At R = 1.5733 Bohr, the simulation yielded:

- Energy: -1.112354 ± 0.003448 Hartree
- Equivalent to: -30.2687 ± 0.0938 eV
- Acceptance rate: 49.71%
- Standard deviation of local energies: 0.3944 Hartree

Block averaging demonstrated consistent results across simulation epochs, indicating adequate equilibration. The statistical uncertainty of ±0.09 eV reflects reasonable sampling efficiency in the equilibrium region.

Error Sources

The primary limitation stems from the choice of wavefunction. Fixed, unoptimized 1s Slater orbitals were used—the ground state of isolated hydrogen atoms. Within a molecule, electrons experience a fundamentally different potential, and the optimal orbital shape should differ accordingly from the atomic case.

A more complete treatment would incorporate:

- Scaling parameters for orbital exponents (variational optimization)
- Explicitly correlated terms to account for electron-electron interactions
- Configuration interaction expansions

The fixed ansatz cannot accommodate any of these refinements, explaining both the elongated bond length prediction and the significant underestimation of binding energy.

The Monte Carlo statistical uncertainty (±0.003 Hartree) is relatively small and not the dominant error source. Instead, the systematic error from the inadequate wavefunction approximation dominates the discrepancies.

Comparison with Other Methods

Method	Bond Length (Å)	Binding Energy (eV)
VMC (this work)	0.8326	3.3256
Hartree-Fock	0.7383	4.747
Experiment	0.7414	4.747

Hartree-Fock substantially outperforms this VMC approach, primarily because HF orbitals undergo self-consistent optimization. In contrast, the present calculation uses a fixed ansatz. Full Configuration Interaction methods achieve near-perfect agreement with experimental values.

Conclusions

The VMC calculation successfully maps a potential energy surface and identifies a bonding minimum, demonstrating the underlying method and algorithm function correctly. However, the quantitative results—the 12% bond length error and 30% binding energy underestimation—illustrate the necessity of flexible, optimizable wavefunctions in modern molecular quantum chemistry.

This study reveals both strengths and limitations: VMC methodology is sound, but wavefunction flexibility directly impacts accuracy. Quantitative improvements would require allowing the orbital exponent to vary or including additional terms in the ansatz. The findings underscore that simple approximations demand careful validation against benchmarks before their predictions can be trusted for real applications.