TC Exercise 2

Nicolás Rojo Esteban

Compare the results for the different algorithms using CO and H2O

In this work, we implement and compare four optimization algorithms—Steepest Descent, AdaGrad, AdaDelta, and Adam—to calculate the orbital energies. All these algorithms share a common iterative procedure, which can be summarized in the following steps:

- Firstly, the occupation numbers of the orbitals and the Natural Orbital Functional (NOF) coefficients are computed.
- Using these quantities, the energy is evaluated through the `calcorbe` function.
- With the calculated energy, gradients with respect to orbital parameters are determined.
- Depending on the chosen optimization method, these gradients are used to update the orbital parameters accordingly.
- This iterative process continues until the convergence criterion is met, specifically when the norm of the gradient is lower than 10^{-3}.

Steepest Descent Algorithm. The Steepest Descent algorithm is a gradient-based optimization method that iteratively searches for a local minimum of a given energy function $E(\theta)$. Starting from an initial guess of the parameters θ_0, the method evaluates the gradient $\nabla E(\theta)$ at the current point and updates the parameters by moving in the opposite direction of this gradient, scaled by a fixed step size (learning rate) α. Formally, each iteration updates the parameters according to:

$$\theta_{t+1} = \theta_t - \alpha \, \nabla E(\theta_t),$$

where $\alpha > 0$ controls how large each optimization step is.

The Steepest Descent algorithm updates the parameters by moving in the direction opposite to the gradient, scaled by a learning rate α. Selecting a suitable value for α is critical, as it directly impacts convergence efficiency. If α is excessively large—for instance, around 0.031, the calculation fails to converge. Conversely, if α is chosen too small, convergence becomes very slow, significantly increasing computational cost.

Table 1: Steepest Descent results for CO and H_2O

Molecule	α	Iterations	Time (s)	Energy
CO	0.01	3292	894.41	-113.007958
H_2O	0.02	1597	358.32	-76.205202
H_2O	0.01	3192	686.26	-76.205201

The learning rate parameter α in the Steepest Descent algorithm determines the step size taken along the gradient direction at each iteration, significantly affecting convergence behavior. A suitable choice of α is crucial: if α is excessively large (e.g., around 0.031 in our calculations), the optimization fails to converge, causing the energy values to oscillate

or diverge indefinitely. Conversely, if α is set too small, the optimization progresses very slowly, substantially increasing computational time. As shown in Table 1, reducing α to 0.01 for CO or H_2O greatly increases the required iterations and computational time to achieve convergence adn the result of the calculation is practically the same.

AdaGrad Algorithm. AdaGrad (Adaptive Gradient Algorithm) is an optimization method that adjusts the learning rate individually for each parameter based on past gradients. At iteration t, given parameters θ_t, AdaGrad updates them according to:

$$\theta_{t+1,i} = \theta_{t,i} - \frac{\alpha}{\sqrt{G_{t,ii} + \varepsilon}} \nabla E(\theta_t)_i,$$

where α is the initial learning rate, and G_t is a diagonal matrix accumulating squared gradients:

$$G_t = G_{t-1} + \nabla E(\theta_t) \odot \nabla E(\theta_t),$$

with $\odot$ indicating element-wise multiplication. The term ε (typically 10^{-8}) is introduced to prevent division by zero:

$$\theta_{t+1,i} = \theta_{t,i} - \frac{\alpha}{\sqrt{G_{t,ii} + \varepsilon}} \nabla E(\theta_{t,i}),$$

This adaptive step size ensures parameters with larger gradients update more conservatively, improving convergence especially in cases with highly uneven gradient distributions.

Table 2: Results obtained with AdaGrad optimization for CO and H_2O using different step sizes α.

Molecule	α	Iterations	Time (s)	Energy
CO	0.01	259	80.38	-113.032873
H_2O	0.01	173	41.828103	-76.237800
H_2O	0.02	61	15.250063	-76.238074

Table 2 presents the outcomes of the AdaGrad optimization for CO and H_2O. The algorithm leverages adaptive step sizes by accumulating the squared gradients for each parameter, leading to a more nuanced update rule compared to standard Steepest Descent. Comparing with the steepest descent, there are big differences in both iteration count and total runtime. For CO with $\alpha = 0.01$, Steepest Descent converges in 3292 iterations over 894.41 seconds, reaching an energy of -113.007958, whereas the equivalent AdaGrad run (also at $\alpha = 0.01$) requires only 259 iterations and 80.38 seconds, achieving a slightly lower energy of -113.032873. Similar behavior occurs in H_2O: with $\alpha = 0.02$, Steepest Descent needs 1597 iterations and 358.32 seconds to converge (-76.205202), whereas AdaGrad converges in 61 iterations (15.25 seconds) and yields -76.238074. These comparisons highlight that AdaGrad not only accelerates convergence but often attains a slightly lower final energy than Steepest Descent for the same or similar learning rates.

AdaDelta Algorithm. AdaDelta is an adaptive learning-rate method that extends AdaGrad by considering both the decaying average of squared gradients and the decaying average of squared parameter updates. At each iteration t, given a parameter vector θ_t and its gradient $\nabla E(\theta_t)$, the algorithm maintains two accumulators: $E[g^2]_t$, the exponential moving average of past squared gradients, and $E[\Delta\theta^2]_t$, the exponential moving average of past squared updates. The gradients are first accumulated as

$$E[g^2]_t = \rho\, E[g^2]_{t-1} \;+\; (1-\rho)\left(\nabla E(\theta_t)\right)^2,$$

where $\rho \in (0,1)$ is a decay parameter, often set near 0.9 or 0.95. The parameter update $\Delta\theta_t$ is then computed as

$$\Delta\theta_t = -\,\frac{\sqrt{E[\Delta\theta^2]_{t-1} \;+\; \varepsilon}}{\sqrt{E[g^2]_t \;+\; \varepsilon}}\,\nabla E(\theta_t),$$

where ε is a small constant (e.g., 10^{-8}) to avoid division by zero. Next, the average of squared parameter updates is refreshed:

$$E[\Delta\theta^2]_t = \rho\, E[\Delta\theta^2]_{t-1} \;+\; (1-\rho)\left(\Delta\theta_t\right)^2.$$

Finally, the parameters are updated as $\theta_{t+1} = \theta_t + \Delta\theta_t$. This approach adapts the effective step size over time without requiring a global learning rate, often leading to more stable and efficient convergence than standard gradient descent.

Table 3: Results obtained with AdaDelta optimization for CO and H_2O.

Molecule	ρ	ε	Iterations	Energy
CO	0.8	1×10^{-10}	187	-113.032097
H_2O	0.8	1×10^{-10}	190	-76.238403

In the calculations performed with AdaGrad algorithm, the gradient norm increases at each iteration rather than decreasing, so the usual convergence criterion based on the gradient norm is never met. However, after a certain point, the energy remains unchanged, indicating that the algorithm has effectively stabilized. We mark this iteration in Table 3 as the point where the energy stops varying.

Unlike methods such as Steepest Descent or AdaGrad, AdaDelta uses an exponential decay parameter ρ to control how quickly previous gradient information is "forgotten." A higher ρ (often in the 0.8–0.95 range) places more emphasis on older gradients, smoothing out updates. Here, we set $\rho = 0.8$, which allowed the energy to settle while the gradient norm continued growing.

ADAM Adam (Adaptive Moment Estimation) is an adaptive gradient method that combines ideas from momentum and RMSProp. At each iteration t, given parameters θ_t and their gradient $\nabla E(\theta_t)$, Adam computes two exponential moving averages: the first moment m_t for the gradient and the second moment v_t for the squared gradient:

$$m_t = \beta_1 m_{t-1} + (1-\beta_1)\,\nabla E(\theta_t), \qquad v_t = \beta_2 v_{t-1} + (1-\beta_2)\left(\nabla E(\theta_t)\right)^2.$$

Both are then bias-corrected:

$$\hat{m}_t = \frac{m_t}{1-\beta_1^t}, \qquad \hat{v}_t = \frac{v_t}{1-\beta_2^t}.$$

Finally, the parameter update is performed as

$$\theta_{t+1} = \theta_t - \alpha \frac{\hat{m}_t}{\sqrt{\hat{v}_t} + \varepsilon},$$

where α is the global learning rate and ε is a small constant (e.g., 10^{-8}) introduced for numerical stability.

Table 4: Results obtained with the Adam optimizer for CO and H_2O.

Molecule	α	β_1	β_2	Iterations	Gradient	Energy
CO	0.01	0.7	0.999	24	7.50×10^{-3}	-113.033599
H_2O	0.01	0.7	0.999	23	5.60×10^{-3}	-76.23763799
H_2O	0.02	0.7	0.999	17	1.20×10^{-2}	-76.23794875

In the calculations for CO and H_2O (see Table 4), neither the gradient norm nor the energy changed significantly after the reported iteration, meaning that the usual convergence criterion was not strictly fulfilled. Nevertheless, Adam quickly stabilized both the energy and the gradient within a very small number of iterations, making it the fastest method among those examined. Although the final gradient remained slightly above the nominal threshold, no further improvement in energy was observed, indicating that Adam had effectively reached a minimum.

Table 5: Results for all algorithms using $\alpha = 0.01$ where applicable.

Molecule	Algorithm	Iterations	Energy
CO	Steepest Descent	3292	-113.007958
CO	AdaGrad	259	-113.0328734
CO	AdaDelta	187	-113.0320972
CO	Adam	24	-113.033599
H_2O	Steepest Descent	3192	-76.2052011
H_2O	AdaGrad	173	-76.23779964
H_2O	AdaDelta	190	-76.23840289
H_2O	Adam	23	-76.23763799

Comparison of Methods. Table 5 summarizes the performance of four optimization algorithms applied to CO and H_2O. Although Steepest Descent eventually converges, it requires by far the largest number of iterations. AdaGrad and AdaDelta significantly reduce the iteration count while achieving energies close to or lower than the Steepest Descent results. Notably, Adam requires the fewest iterations, converging in under 30 steps for both molecules while reaching energies comparable to (or slightly better than) the other methods. This makes Adam the fastest approach among those tested.

As shown in Table 5, Steepest Descent consistently yields a slightly higher final energy compared to the other three methods. By contrast, AdaGrad, AdaDelta, and Adam converge to very similar (and lower) energy values. Among these, Adam stands out for reaching its final energy in the fewest iterations, making it the most efficient of the four methods tested. Consequently, while the three adaptive methods outperform Steepest Descent in terms of final energy, Adam is the clear winner, combining rapid convergence with an equally accurate or marginally better final energy.

Finally, choose the optimizer that provided you the best results. Could you improve the performance of that optimizer playing dynamically with the values of alpha?

To further enhance Adam, we introduce a simple adjustment mechanism that reduces the learning rate α if no improvement in energy is observed and allows additional optimization steps:

```
if not improved:
    p.alpha = p.alpha / 10
    p.maxloop = p.maxloop + 30
```

Whenever the algorithm finds that the energy has not decreased relative to the best value found so far (`not improved`), the code decreases α by a factor of 10, making the updates more conservative. Simultaneously, it increases (the maximum number of iterations) by 30, giving the optimizer more room to refine the solution at a smaller step size.

Table 6: Comparison of Adam with and without dynamic α adjustment.

Molecule	Algorithm	Iterations	Energy
CO	Adam	24	-113.033599
CO	Adam + dynamic α	24	-113.0336003
H_2O	Adam	23	-76.23763799
H_2O	Adam + dynamic α	23	-76.23763891

Although the final energies remain practically identical to those achieved without dynamic α adjustment, this strategy ensures that the formal gradient-based convergence criterion is reached, rather than simply halting when the energy ceases to improve.