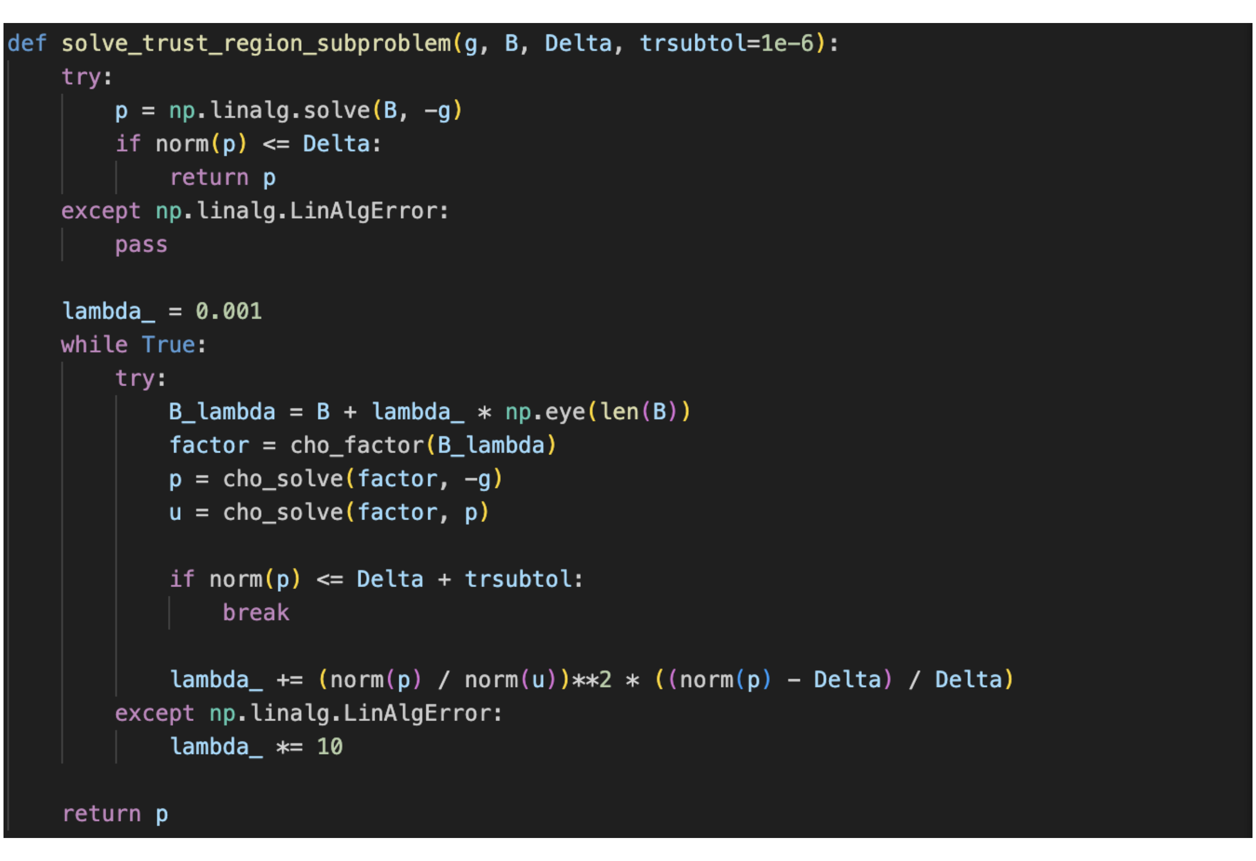
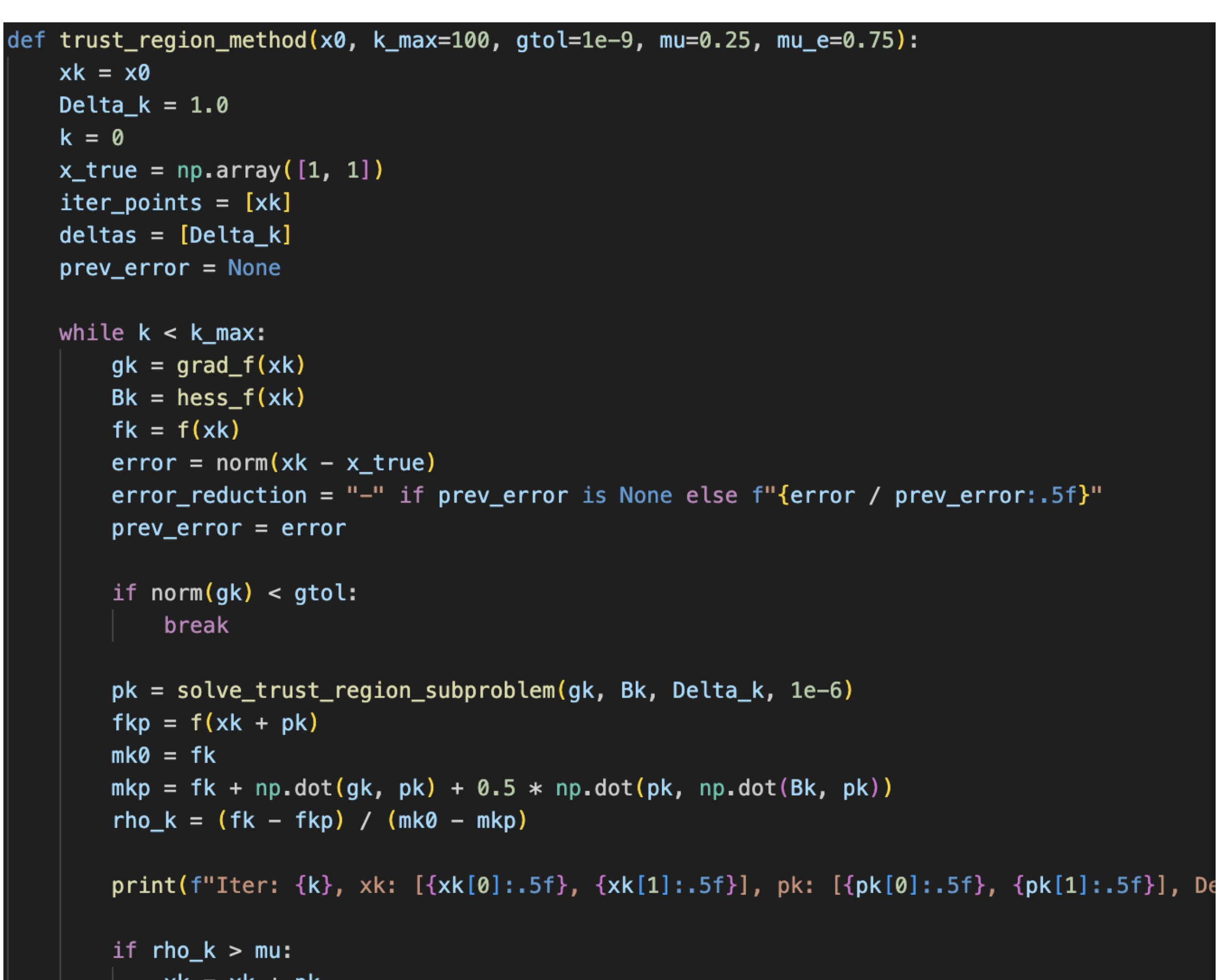
MATH 140: Mathematical Methods for Optimization
Assignment 7—Spring 2024
Due April 4th, 2024
By:Ronald Nap

1. (**20 points**, the trust-region method)

   (a) Implement the trust-region method in matlab or python (algorithms 1 and 2 listed on page 2).



```
def solve_trust_region_subproblem(g, B, Delta, trsubtol=1e-6):
    try:
        p = np.linalg.solve(B, -g)
        if norm(p) <= Delta:
            return p
    except np.linalg.LinAlgError:
        pass

    lambda_ = 0.001
    while True:
        try:
            B_lambda = B + lambda_ * np.eye(len(B))
            factor = cho_factor(B_lambda)
            p = cho_solve(factor, -g)
            u = cho_solve(factor, p)

            if norm(p) <= Delta + trsubtol:
                break

            lambda_ += (norm(p) / norm(u))**2 * ((norm(p) - Delta) / Delta)
        except np.linalg.LinAlgError:
            lambda_ *= 10

    return p
```

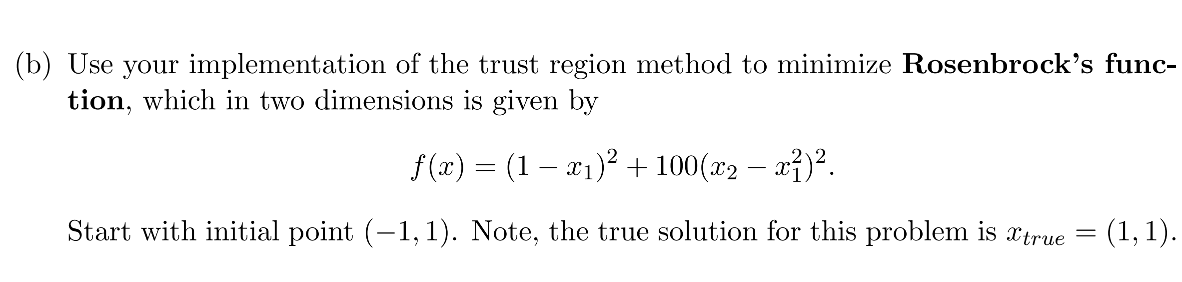


```
def trust_region_method(x0, k_max=100, gtol=1e-9, mu=0.25, mu_e=0.75):
    xk = x0
    Delta_k = 1.0
    k = 0
    x_true = np.array([1, 1])
    iter_points = [xk]
    deltas = [Delta_k]
    prev_error = None

    while k < k_max:
        gk = grad_f(xk)
        Bk = hess_f(xk)
        fk = f(xk)
        error = norm(xk - x_true)
        error_reduction = "-" if prev_error is None else f"{error / prev_error:.5f}"
        prev_error = error

        if norm(gk) < gtol:
            break

        pk = solve_trust_region_subproblem(gk, Bk, Delta_k, 1e-6)
        fkp = f(xk + pk)
        mk0 = fk
        mkp = fk + np.dot(gk, pk) + 0.5 * np.dot(pk, np.dot(Bk, pk))
        rho_k = (fk - fkp) / (mk0 - mkp)

        print(f"Iter: {k}, xk: [{xk[0]:.5f}, {xk[1]:.5f}], pk: [{pk[0]:.5f}, {pk[1]:.5f}], De

        if rho_k > mu:
            xk = xk + pk
            iter_points.append(xk)
            deltas.append(Delta_k)
            if rho_k >= mu_e:
                Delta_k = max(Delta_k, 2*norm(pk))
        else:
            Delta_k = Delta_k / 2

        k += 1

    return xk, k, iter_points, deltas
```

   (b) Use your implementation of the trust region method to minimize **Rosenbrock's function**, which in two dimensions is given by

$$f(x) = (1-x_1)^2 + 100(x_2 - x_1^2)^2.$$

   Start with initial point $(-1, 1)$. Note, the true solution for this problem is $x_{true} = (1, 1)$.



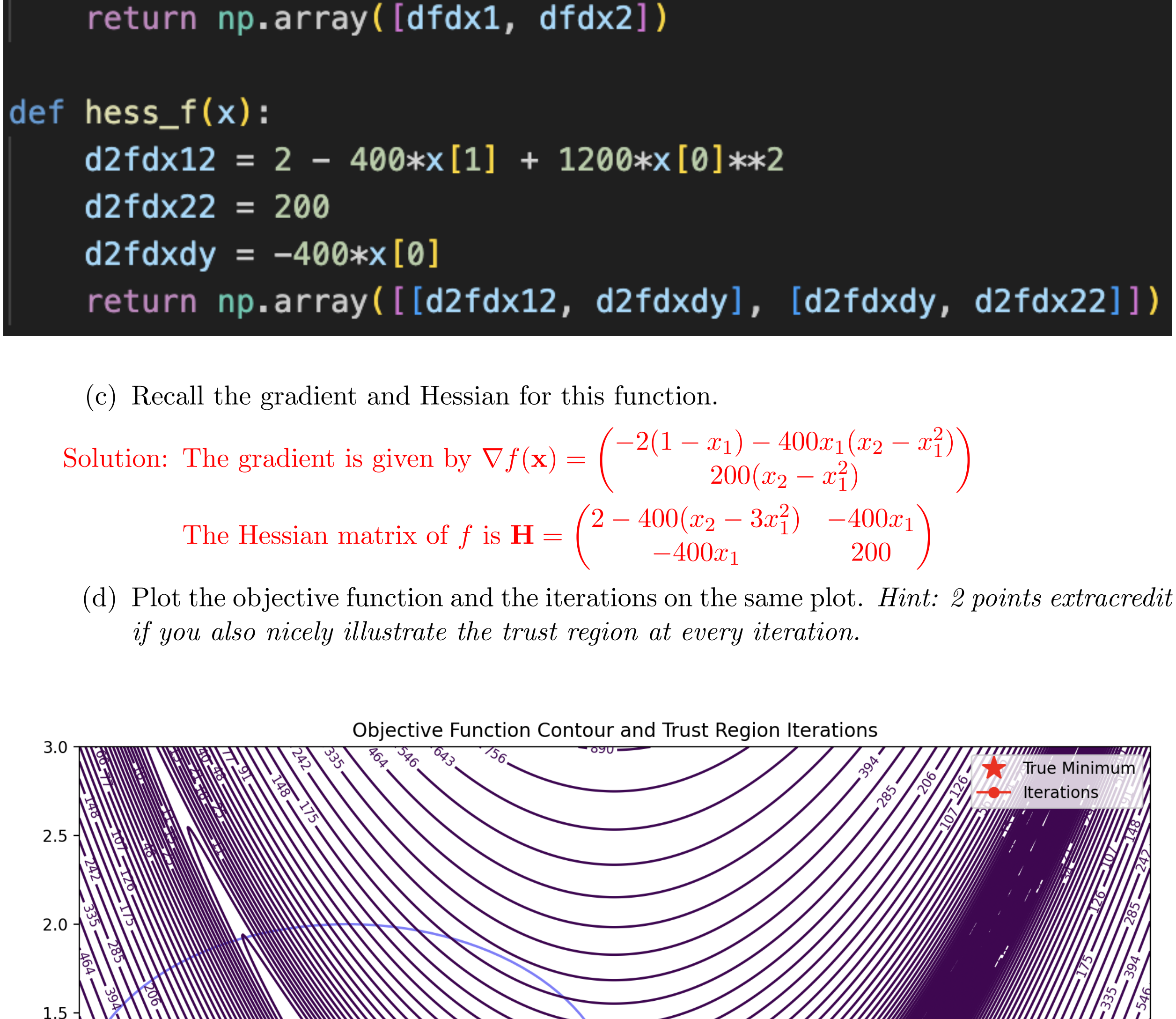
```
def f(x):
    return (1 - x[0])**2 + 100*(x[1] - x[0]**2)**2

def grad_f(x):
    dfdx1 = -2*(1 - x[0]) - 400*x[0]*(x[1] - x[0]**2)
    dfdx2 = 200*(x[1] - x[0]**2)
    return np.array([dfdx1, dfdx2])

def hess_f(x):
    d2fdx12 = 2 - 400*x[1] + 1200*x[0]**2
    d2fdx22 = 200
    d2fdxdy = -400*x[0]
    return np.array([[d2fdx12, d2fdxdy], [d2fdxdy, d2fdx22]])
```

   (c) Recall the gradient and Hessian for this function.

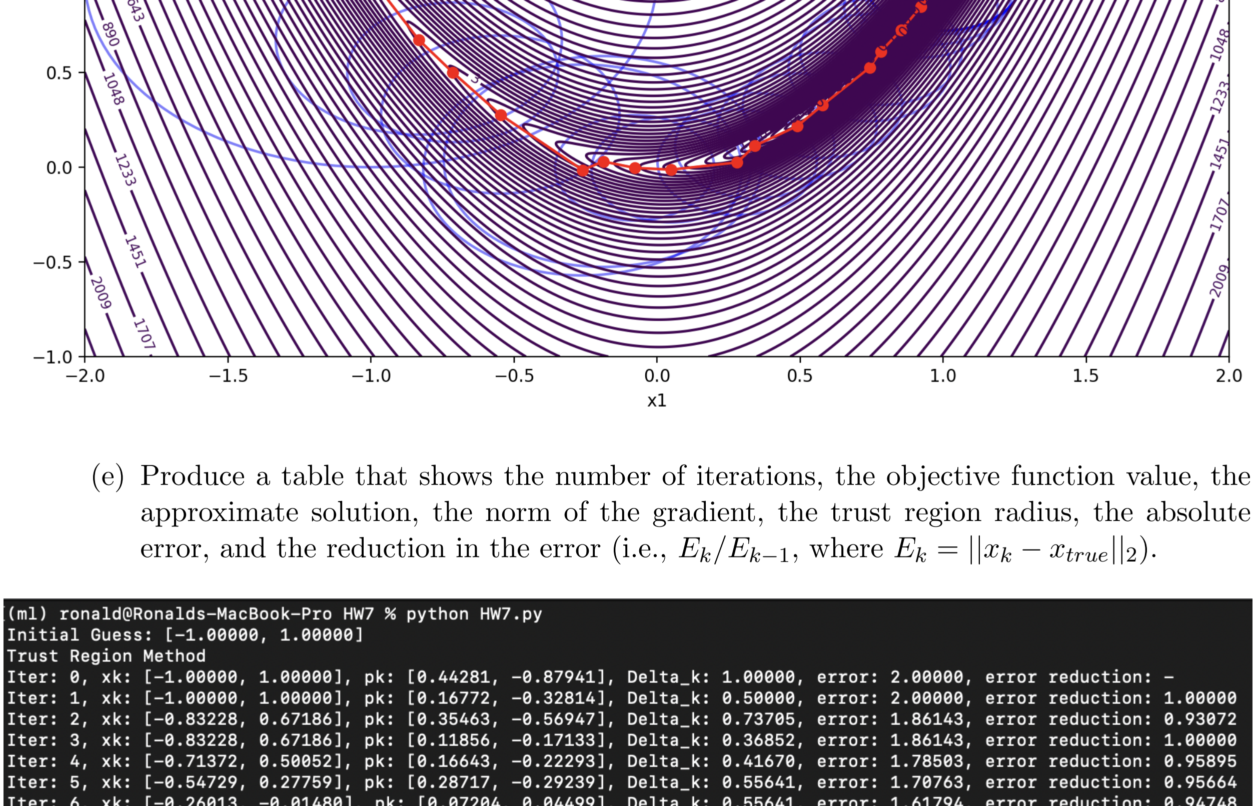
Solution: The gradient is given by $\nabla f(\mathbf{x}) = \begin{pmatrix} -2(1-x_1) - 400x_1(x_2 - x_1^2) \\ 200(x_2 - x_1^2) \end{pmatrix}$

The Hessian matrix of f is $\mathbf{H} = \begin{pmatrix} 2 - 400(x_2 - 3x_1^2) & -400x_1 \\ -400x_1 & 200 \end{pmatrix}$

   (d) Plot the objective function and the iterations on the same plot. *Hint: 2 points extracredit if you also nicely illustrate the trust region at every iteration.*

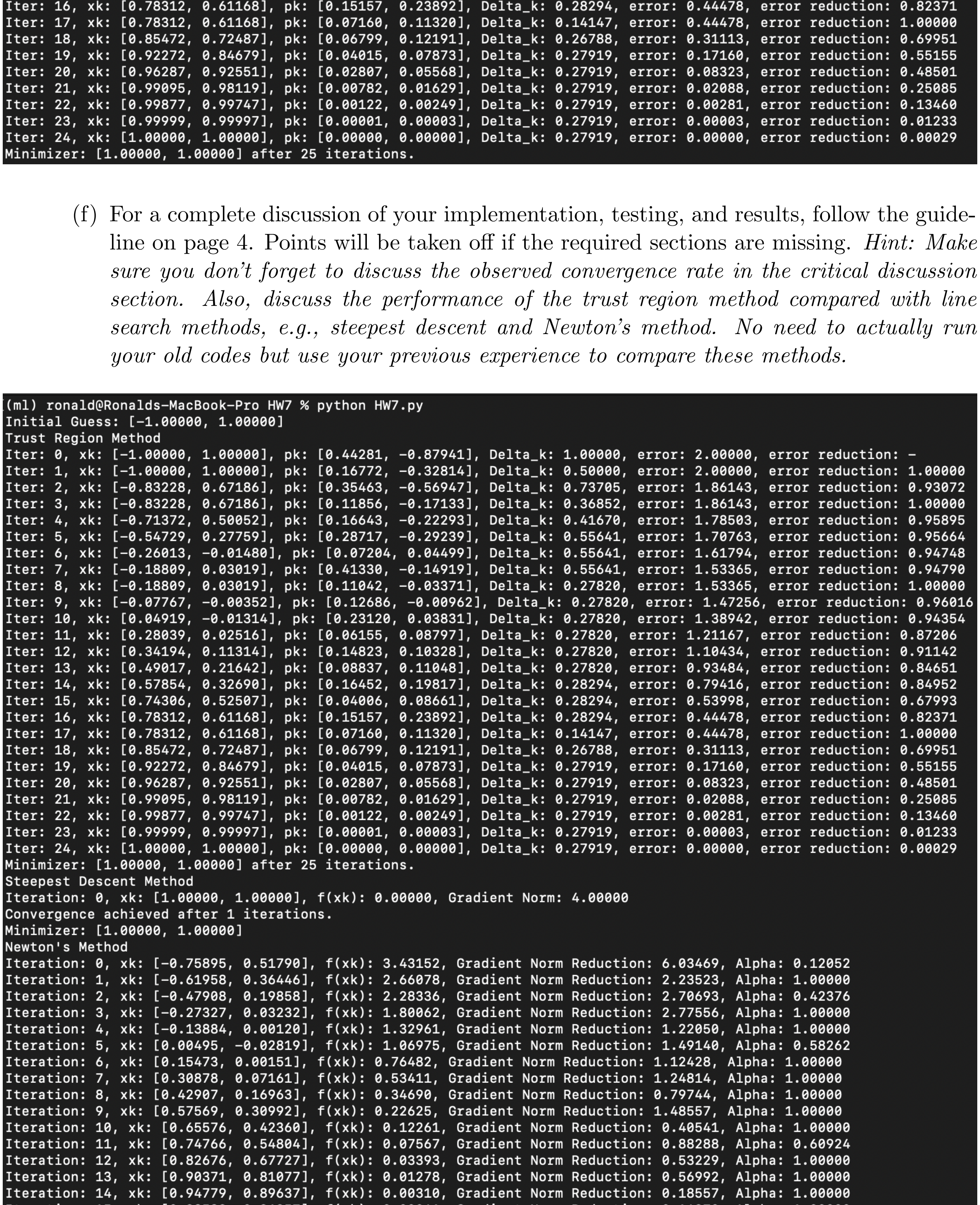


   (e) Produce a table that shows the number of iterations, the objective function value, the approximate solution, the norm of the gradient, the trust region radius, the absolute error, and the reduction in the error (i.e., E_k/E_{k-1}, where $E_k = ||x_k - x_{true}||_2$).



```
(ml) ronald@Ronalds-MacBook-Pro HW7 % python HW7.py
Initial Guess: [-1.00000, 1.00000]
Trust Region Method
Iter: 0, xk: [-1.00000, 1.00000], pk: [0.44281, -0.87941], Delta_k: 1.00000, error: 2.00000, error reduction: -
Iter: 1, xk: [-1.00000, 1.00000], pk: [0.16772, -0.32814], Delta_k: 0.50000, error: 2.00000, error reduction: 1.00000
Iter: 2, xk: [-0.83228, 0.67186], pk: [0.35463, -0.56947], Delta_k: 0.73705, error: 1.86143, error reduction: 0.93072
Iter: 3, xk: [-0.83228, 0.67186], pk: [0.11856, -0.17133], Delta_k: 0.36852, error: 1.86143, error reduction: 1.00000
Iter: 4, xk: [-0.71372, 0.50052], pk: [0.16643, -0.22293], Delta_k: 0.41670, error: 1.78503, error reduction: 0.95895
Iter: 5, xk: [-0.54729, 0.27759], pk: [0.28717, -0.29239], Delta_k: 0.55641, error: 1.70763, error reduction: 0.95664
Iter: 6, xk: [-0.26013, -0.01480], pk: [0.07204, 0.04499], Delta_k: 0.55641, error: 1.61794, error reduction: 0.94748
Iter: 7, xk: [-0.18809, 0.03019], pk: [0.41330, -0.14919], Delta_k: 0.55641, error: 1.53365, error reduction: 0.94790
Iter: 8, xk: [-0.18809, 0.03019], pk: [0.11042, -0.03371], Delta_k: 0.27820, error: 1.53365, error reduction: 1.00000
Iter: 9, xk: [-0.07767, -0.00352], pk: [0.12686, -0.00962], Delta_k: 0.27820, error: 1.47256, error reduction: 0.96016
Iter: 10, xk: [0.04919, -0.01314], pk: [0.23120, 0.03831], Delta_k: 0.27820, error: 1.38942, error reduction: 0.94354
Iter: 11, xk: [0.28039, 0.02516], pk: [0.06155, 0.08797], Delta_k: 0.27820, error: 1.21167, error reduction: 0.87206
Iter: 12, xk: [0.34194, 0.11314], pk: [0.14823, 0.10328], Delta_k: 0.27820, error: 1.10434, error reduction: 0.91142
Iter: 13, xk: [0.49017, 0.21642], pk: [0.08837, 0.11048], Delta_k: 0.27820, error: 0.93484, error reduction: 0.84651
Iter: 14, xk: [0.57854, 0.32690], pk: [0.16452, 0.19817], Delta_k: 0.28294, error: 0.79416, error reduction: 0.84952
Iter: 15, xk: [0.74306, 0.52507], pk: [0.04006, 0.08661], Delta_k: 0.28294, error: 0.53998, error reduction: 0.67993
Iter: 16, xk: [0.78312, 0.61168], pk: [0.15157, 0.23892], Delta_k: 0.28294, error: 0.44478, error reduction: 0.82371
Iter: 17, xk: [0.78312, 0.61168], pk: [0.07160, 0.11320], Delta_k: 0.14147, error: 0.44478, error reduction: 1.00000
Iter: 18, xk: [0.85472, 0.72487], pk: [0.06799, 0.12191], Delta_k: 0.26788, error: 0.31113, error reduction: 0.69951
Iter: 19, xk: [0.92272, 0.84679], pk: [0.04015, 0.07873], Delta_k: 0.27919, error: 0.17160, error reduction: 0.55155
Iter: 20, xk: [0.96287, 0.92551], pk: [0.02807, 0.05568], Delta_k: 0.27919, error: 0.08323, error reduction: 0.48501
Iter: 21, xk: [0.99095, 0.98119], pk: [0.00782, 0.01629], Delta_k: 0.27919, error: 0.02088, error reduction: 0.25085
Iter: 22, xk: [0.99877, 0.99747], pk: [0.00122, 0.00249], Delta_k: 0.27919, error: 0.00281, error reduction: 0.13460
Iter: 23, xk: [0.99999, 0.99997], pk: [0.00001, 0.00003], Delta_k: 0.27919, error: 0.00003, error reduction: 0.01233
Iter: 24, xk: [1.00000, 1.00000], pk: [0.00000, 0.00000], Delta_k: 0.27919, error: 0.00000, error reduction: 0.00029
Minimizer: [1.00000, 1.00000] after 25 iterations.
```

   (f) For a complete discussion of your implementation, testing, and results, follow the guideline on page 4. Points will be taken off if the required sections are missing. *Hint: Make sure you don't forget to discuss the observed convergence rate in the critical discussion section. Also, discuss the performance of the trust region method compared with line search methods, e.g., steepest descent and Newton's method. No need to actually run your old codes but use your previous experience to compare these methods.*



```
(ml) ronald@Ronalds-MacBook-Pro HW7 % python HW7.py
Initial Guess: [-1.00000, 1.00000]
Trust Region Method
Iter: 0, xk: [-1.00000, 1.00000], pk: [0.44281, -0.87941], Delta_k: 1.00000, error: 2.00000, error reduction: -
Iter: 1, xk: [-1.00000, 1.00000], pk: [0.16772, -0.32814], Delta_k: 0.50000, error: 2.00000, error reduction: 1.00000
Iter: 2, xk: [-0.83228, 0.67186], pk: [0.35463, -0.56947], Delta_k: 0.73705, error: 1.86143, error reduction: 0.93072
Iter: 3, xk: [-0.83228, 0.67186], pk: [0.11856, -0.17133], Delta_k: 0.36852, error: 1.86143, error reduction: 1.00000
Iter: 4, xk: [-0.71372, 0.50052], pk: [0.16643, -0.22293], Delta_k: 0.41670, error: 1.78503, error reduction: 0.95895
Iter: 5, xk: [-0.54729, 0.27759], pk: [0.28717, -0.29239], Delta_k: 0.55641, error: 1.70763, error reduction: 0.95664
Iter: 6, xk: [-0.26013, -0.01480], pk: [0.07204, 0.04499], Delta_k: 0.55641, error: 1.61794, error reduction: 0.94748
Iter: 7, xk: [-0.18809, 0.03019], pk: [0.41330, -0.14919], Delta_k: 0.55641, error: 1.53365, error reduction: 0.94790
Iter: 8, xk: [-0.18809, 0.03019], pk: [0.11042, -0.03371], Delta_k: 0.27820, error: 1.53365, error reduction: 1.00000
Iter: 9, xk: [-0.07767, -0.00352], pk: [0.12686, -0.00962], Delta_k: 0.27820, error: 1.47256, error reduction: 0.96016
Iter: 10, xk: [0.04919, -0.01314], pk: [0.23120, 0.03831], Delta_k: 0.27820, error: 1.38942, error reduction: 0.94354
Iter: 11, xk: [0.28039, 0.02516], pk: [0.06155, 0.08797], Delta_k: 0.27820, error: 1.21167, error reduction: 0.87206
Iter: 12, xk: [0.34194, 0.11314], pk: [0.14823, 0.10328], Delta_k: 0.27820, error: 1.10434, error reduction: 0.91142
Iter: 13, xk: [0.49017, 0.21642], pk: [0.08837, 0.11048], Delta_k: 0.27820, error: 0.93484, error reduction: 0.84651
Iter: 14, xk: [0.57854, 0.32690], pk: [0.16452, 0.19817], Delta_k: 0.28294, error: 0.79416, error reduction: 0.84952
Iter: 15, xk: [0.74306, 0.52507], pk: [0.04006, 0.08661], Delta_k: 0.28294, error: 0.53998, error reduction: 0.67993
Iter: 16, xk: [0.78312, 0.61168], pk: [0.15157, 0.23892], Delta_k: 0.28294, error: 0.44478, error reduction: 0.82371
Iter: 17, xk: [0.78312, 0.61168], pk: [0.07160, 0.11320], Delta_k: 0.14147, error: 0.44478, error reduction: 1.00000
Iter: 18, xk: [0.85472, 0.72487], pk: [0.06799, 0.12191], Delta_k: 0.26788, error: 0.31113, error reduction: 0.69951
Iter: 19, xk: [0.92272, 0.84679], pk: [0.04015, 0.07873], Delta_k: 0.27919, error: 0.17160, error reduction: 0.55155
Iter: 20, xk: [0.96287, 0.92551], pk: [0.02807, 0.05568], Delta_k: 0.27919, error: 0.08323, error reduction: 0.48501
Iter: 21, xk: [0.99095, 0.98119], pk: [0.00782, 0.01629], Delta_k: 0.27919, error: 0.02088, error reduction: 0.25085
Iter: 22, xk: [0.99877, 0.99747], pk: [0.00122, 0.00249], Delta_k: 0.27919, error: 0.00281, error reduction: 0.13460
Iter: 23, xk: [0.99999, 0.99997], pk: [0.00001, 0.00003], Delta_k: 0.27919, error: 0.00003, error reduction: 0.01233
Iter: 24, xk: [1.00000, 1.00000], pk: [0.00000, 0.00000], Delta_k: 0.27919, error: 0.00000, error reduction: 0.00029
Minimizer: [1.00000, 1.00000] after 25 iterations.
Steepest Descent Method
Iteration: 0, xk: [1.00000, 1.00000], f(xk): 0.00000, Gradient Norm: 4.00000
Convergence achieved after 1 iterations.
Minimizer: [1.00000, 1.00000]
Newton's Method
Iteration: 0, xk: [-0.75895, 0.51790], f(xk): 3.43152, Gradient Norm Reduction: 6.03469, Alpha: 0.12052
Iteration: 1, xk: [-0.61958, 0.36446], f(xk): 2.66078, Gradient Norm Reduction: 2.23523, Alpha: 1.00000
Iteration: 2, xk: [-0.47908, 0.19858], f(xk): 2.28336, Gradient Norm Reduction: 2.70693, Alpha: 0.42376
Iteration: 3, xk: [-0.27327, 0.03232], f(xk): 1.80062, Gradient Norm Reduction: 2.77556, Alpha: 1.00000
Iteration: 4, xk: [-0.13884, 0.00120], f(xk): 1.32961, Gradient Norm Reduction: 1.22050, Alpha: 1.00000
Iteration: 5, xk: [0.00495, -0.02819], f(xk): 1.06975, Gradient Norm Reduction: 1.49140, Alpha: 0.58262
Iteration: 6, xk: [0.15473, 0.00151], f(xk): 0.76482, Gradient Norm Reduction: 1.12428, Alpha: 1.00000
Iteration: 7, xk: [0.30878, 0.07161], f(xk): 0.53411, Gradient Norm Reduction: 1.24814, Alpha: 1.00000
Iteration: 8, xk: [0.42907, 0.16963], f(xk): 0.34690, Gradient Norm Reduction: 0.79744, Alpha: 1.00000
Iteration: 9, xk: [0.57569, 0.30992], f(xk): 0.22625, Gradient Norm Reduction: 1.48557, Alpha: 1.00000
Iteration: 10, xk: [0.65576, 0.42360], f(xk): 0.12261, Gradient Norm Reduction: 0.40541, Alpha: 1.00000
Iteration: 11, xk: [0.74766, 0.54804], f(xk): 0.07567, Gradient Norm Reduction: 0.88288, Alpha: 0.60924
Iteration: 12, xk: [0.82676, 0.67727], f(xk): 0.03393, Gradient Norm Reduction: 0.53229, Alpha: 1.00000
Iteration: 13, xk: [0.90371, 0.81077], f(xk): 0.01278, Gradient Norm Reduction: 0.56992, Alpha: 1.00000
Iteration: 14, xk: [0.94779, 0.89637], f(xk): 0.00310, Gradient Norm Reduction: 0.18557, Alpha: 1.00000
Iteration: 15, xk: [0.98539, 0.96957], f(xk): 0.00041, Gradient Norm Reduction: 0.14970, Alpha: 1.00000
Iteration: 16, xk: [0.99678, 0.99344], f(xk): 0.00001, Gradient Norm Reduction: 0.01305, Alpha: 1.00000
Iteration: 17, xk: [0.99992, 0.99983], f(xk): 0.00000, Gradient Norm Reduction: 0.00107, Alpha: 1.00000
Iteration: 18, xk: [1.00000, 1.00000], f(xk): 0.00000, Gradient Norm Reduction: 0.00000, Alpha: 1.00000
Iteration: 19, xk: [1.00000, 1.00000], f(xk): 0.00000, Gradient Norm Reduction: 0.00000, Alpha: 1.00000
Convergence achieved after 20 iterations.
Minimizer: [1.00000, 1.00000]
BFGS Method
Iteration: 0, xk: [1.00000, 1.00000], f(xk): 0.00000, Gradient Norm Reduction: 0.00000, Alpha: 0.50000
Convergence achieved after 1 iterations.
Minimizer: [1.00000, 1.00000]
```

Solution: To minimize Rosenbrock's function, we compare the results of different optimization techniques learned in class so far: the trust region method, the steepest descent method, and Newton's method. The trust region method leverages both the gradient and Hessian of the function to optimize within a "trust" region around the current point. The steepest descent method takes steps proportional to the negative gradient, with convergence depending on the step size in the steepest descent direction. Newton's method utilizes second-order information by involving the Hessian matrix, aiming for a quadratic convergence rate near the minimum, theoretically outpacing steepest descent. Similarly, the BFGS method approximates Newton's method without directly computing the Hessian, instead updating an estimate of its inverse at each iteration. The initial guess was set to $[-1, 1]$, and the results show the trust region method converging to the known minimum $[1, 1]$ in 25 iterations. Newton's method converges to $[1, 1]$ in 19 iterations. Intriguingly, the steepest descent and BFGS methods indicated convergence in one iteration. However, further tests with a variety of initial guesses suggest that this swift convergence is not a feature of the algorithms themselves but rather a result of the starting point's influence. The trust region method's observed convergence aligns with theoretical expectations, converging reliably but not as swiftly as Newton's method, which is anticipated due to the latter's quadratic convergence rate.

Algorithm 1: Trust-Region Method

```
Set maximum iteration $k_{\max}$, gtol, trsubtol $0 < \mu < \mu_e < 1$;
$k \leftarrow 0$, $\Delta_k \leftarrow 1$, $x_k \leftarrow x_0$;
while $k \le k_{\max}$ and $\|g_k\|_2 >=$ gtol
    Obtain $p_k$ by solving the TR subproblem (implement Algorithm 2)
    Evaluate $\rho_k = [f(x_k) - f(x_k + p_k)]/[m_k(0) - m_k(p_k)]$;
    if $\rho_k > \mu$, then
        $x_{k+1} = x_k + p_k$;
        if $\rho_k \ge \mu_e$ then
            $\Delta_{k+1} = \max\{\Delta_k, 2\|p_k\|_2\}$;
        else
            $\Delta_{k+1} = \Delta_k$;
        end
    else
        $x_{k+1} \leftarrow x_k$;
        $\Delta_{k+1} \leftarrow \frac{1}{2}\|p_k\|_2$;
    end
    $k \leftarrow k + 1$;
end
```

Algorithm 2: Solving the Trust-Region Subproblem

```
Let $\Delta > 0$;
if $B$ is positive definite
    Solve $Bp = -g$
    if $\|p\|_2 \le \Delta$, then return $p$
end
Let $\lambda \ge 0$ with $B + \lambda I$ positive definite;
while not converged
    Factor $B + \lambda I = R^T R$;        Use Cholesky factorization
    Solve $R^T R p = -g$;
    Solve $R^T u = p$;
    $\lambda \leftarrow \lambda + \left(\frac{\|p\|_2}{\|u\|_2}\right)^2 \left(\frac{\|p\|_2 - \Delta}{\Delta}\right)$
end
```

Some clarifications and practical choices for the parameters present in the TR algorithm:

1. Choose B as the Hessian of the objective function $f(x)$.
2. For the subproblem, `not converged` means
   `while abs(norm(p)-Delta) > trsubtol && k < kmax,`
   where `trsubtol`, `k` the current iteration, and `kmax` is the max number of iterations.
3. $\mu_e = \frac{3}{4}$, $\mu = \frac{1}{4}$, gtol $= 10^{-9}$, trsubtol $= 10^{-6}$.

Tips on how to report on computer results[1].

In mathematics, as in all sciences, one of the challenges is to learn how to communicate our findings. Therefore, I would like to guide you to learn how to explain clearly and in a professional style what you did and to present and analyze your results. Therefore, for this take home exam, please follow the **outline** given below for your report. (I would recommend this outline for any homework problem that involves computer code.)

Section 1: Problem Description

State the mathematical problem, e.g., "we want to solve the following optimization problem ...", and state in brief words the name of the numerical method used, e.g., "... using line search or trust region, etc." (without stating formulas).

Section 2: Numerical Method Implemented

Explain your numerical method in detail, with suitable formulas; you might be able to just quote a proper page of the textbook (or recommended book), etc. This can be short if the algorithm is stated fully somewhere (as is the case here, since the algorithms are on page 3), but this section may be a major part of your report, if you have to collect a lot of information from various sources or have to derive all equations yourself. Then you should discuss the algorithm a little, e.g., how it works, what theorems apply, what convergence behavior you expect in which quantity, and why you believe this might work. Here I would summarise the method as you understood it. Give a detailed description of how you tested your code.

Section 3: Computational Experiments and Results

Describe the computational experiments performed; that is, state exactly the values of all parameters and other values that have not been specified, yet. Then, introduce your results by explaining how they are presented; you must explicitly refer to every figure or table that you include and introduce each function plotted or column in table. The point is to define what your labels in the figures and tables mean. Use formulas as necessary to define quantities clearly. You should mention results concretely, e.g., for an error plot, you might observe that "the absolute value of the error is never larger than ...".

Section 4: Critical Discussion of Results

In this section you want to contrast your results with applicable theorems or compare results from several cases with each other. Notice that this section is often very short, namely if you set things up well in Section 2 (theorems quoted and convergence expectation specified) and then plot or print out exactly these quantities in Section 3.

References

Provide the complete bibliographic information of your references here if and when applicable.

---

[1] These tips (with slight modifications) are from "How to Report on Computer Results" by Dr. Matthias Gobbert, http://userpages.umbc.edu/~gobbert/teaching/math441.20158/writeup.html