Numerical Optimization, Fall 2024
Homework 5

Due 23:59 (CST), Nov. 24, 2024

1. Prove that by applying the quasi-Newton curvature condition $s_k^T y_k > 0$ in the univariate case, the quasi-Newton method is equivalent to the secant method.[10pts]

   Solution:
   From the update of Quasi-Newton method
   $$x_{k+2} = x_{k+1} - H_{k+1}^{-1} f^{'}(x_{k+1}).$$
   Then, due to the secant equation $H_{k+1}s_k = y_k$, then
   $$H_{k+1}^{-1} = \frac{s_k}{y_k} = \frac{x_{k+1} - x_k}{f^{'}(x_{k+1}) - f^{'}(x_k)}.$$
   Therefore,
   $$x_{k+2} = x_{k+1} - \frac{x_{k+1} - x_k}{f^{'}(x_{k+1}) - f^{'}(x_k)} f^{'}(x_{k+1}). \quad \text{(Secant Method)}$$

2. Consider an unconstrained optimization problem: $\min_x f(x)$, where $f(x) \in \mathcal{C}^2$ and $f(x)$ is L-smooth on a bounded level set $\{x | f(x) \leq f(x_0)\}$. Prove the convergence of backtracking Amijo line search.[10pts]

   Solution:
   According to Taylor's Theorem, there exists $t \in (0, 1)$ such that
   $$f(x_k + \alpha_k d_k) = f(x_k) + \alpha_k \nabla f(x_k)^T d_k + \frac{1}{2}\alpha_k^2 d_k^T \nabla_k^2 f(x_k + t\alpha_k d_k)^T d_k,$$
   since $f(x)$ is L-smooth,
   $$||\nabla^2 f(x)|| \leq L,$$
   Thus,
   $$f(x_k + \alpha_k d_k) \leq f(x_k) + \alpha_k \nabla f(x_k)^T d_k + \frac{L}{2}\alpha_k^2 ||d_k||_2^2,$$
   The Armijo condition is given by:
   $$f(x_k + \alpha d_k) \leq f(x_k) + c_1 \alpha \nabla f(x_k)^T d_k.$$

Backtracking Amijo line search reduces α_k geometrically ($\alpha_k = \beta^k \alpha_0, \beta \in (0,1)$) until $f(x_k)+c_1\alpha_k \nabla f(x_k)^T d_k \leq f(x_k)+\alpha_k \nabla f(x_k)^T d_k + \frac{L}{2}\alpha^2 ||d_k||_2^2$, i.e., $\alpha_k \leq \frac{2(c_1-1)\nabla f(x_k)^T d_k}{L||d_k||^2}$ to satisfy the Armijo condition.
Now we have

$$\begin{aligned} f(x_k) - f(x_{k+1}) &\geq -c_1\alpha_k \nabla f(x_k)^T d_k \\ &\geq -c_1 \nabla f(x_k)^T d_k \frac{2(c_1-1)\nabla f(x_k)^T d_k}{L||d_k||^2} \\ &\geq \frac{2c_1(1-c_1)}{L}||\nabla f(x_k)||^2, \end{aligned}$$

thus,

$$||\nabla f(x_k)||^2 \leq \frac{L}{2c_1(1-c_1)}(f(x_k) - f(x_{k+1})).$$

Summing from $k=0$ to $K-1$,

$$\begin{aligned} \frac{1}{K}\sum_{k=0}^{K-1} \|\nabla f(x_k)\|^2 &\leq \frac{L}{2c_1(1-c_1)K}\sum_{k=0}^{K-1}(f(x_k) - f(x_{k+1})) \\ &= \frac{L}{2c_1(1-c_1)K}(f(x_0) - f(x_K)) \\ &\leq \frac{L}{2c_1(1-c_1)K}(f(x_0) - f(x^*)) \\ &\xrightarrow{K\to\infty} 0. \end{aligned}$$

3. For minimizing a quadratic function $f(x)$, $f(x)$ is strictly convex. Given a current point x_k and a descent direction p_k, compute the exact line search steplength, i.e., $\min_\alpha f(x_k + \alpha p_k)$.[20pts]

   Solution:
   Given a quadratic function $f(x) = \frac{1}{2}x^T A x + b^T x + c$, where $A \succ 0$.

   $$\phi(\alpha) = f(x_k + \alpha p_k) = \frac{1}{2}(x_k + \alpha p_k)^T A (x_k + \alpha p_k) + b^T(x_k + \alpha p_k) + c.$$

   Let $\phi^{'}(\alpha) = \alpha p_k^T A p_k + (x_k^T A p_k + b^T p_k) = 0$, we can get $\alpha = -\frac{x_k^T A p_k + b^T p_k}{p_k^T A p_k}$.

4. The Newton direction is $p_k = -\nabla^2 f(x_k)^{-1}\nabla f(x_k)$. When is the Newton direction a descent direction? How do you modify the Newton direction as small as possible to generate a descent direction when the Newton direction is not a descent direction? [20pts]

   Solution:
   The Newton direction p_k is a descent direction if:

   $$\nabla f(x_k)^T p_k < 0.$$

Substituting p_k:

$$\nabla f(x_k)^T(-\nabla^2 f(x_k)^{-1}\nabla f(x_k)) = -\nabla f(x_k)^T\nabla^2 f(x_k)^{-1}\nabla f(x_k).$$

This holds if $\nabla^2 f(x_k)$ is positive definite in the gradient direction.
If $\nabla^2 f(x_k)$ is not positive definite in the gradient direction, modify $\nabla^2 f(x_k)$ by adding a multiple of the identity matrix λI such that $\nabla^2 f(x_k) + \lambda I$ is positive definite in the gradient direction.

5. Derive the simplified Conjugate Gradient (CG) iteration formula(see page34 of lec11).[20pts]

Solution:
Applying $p_k = -r_k + \beta_k p_{k-1}$ and $r_k^T p_i = 0$ $(i = 0, 1, \cdots, k-1)$, we can get $r_k^T p_k = r_k^T(-r_k + \beta_k p_{k-1}) = -r_k^T r_k + \beta_k r_k^T p_{k-1} = -r_k^T r_k$.
Thus,

$$\alpha_k = -\frac{r_k^T p_k}{p_k^T A p_k} = \frac{r_k^T r_k}{p_k^T A p_k}.$$

From $r_k = Ax_k - b$ and $x_{k+1} = x_k + \alpha_k p_k$,
we can get

$$r_{k+1} = Ax_{k+1} - b = A(x_k + \alpha_k p_k) - b = r_k + \alpha_k A p_k.$$

Applying $p_k = -r_k + \beta_k p_{k-1}$ and $r_k^T p_i = 0$ $(i = 0, 1, \cdots, k-1)$ once again, we can get

$$\beta_{k+1} = \frac{r_{k+1}^T A p_k}{p_k^T A p_k} = \frac{r_{k+1}^T (r_{k+1} - r_k)}{p_k^T A p_k}\frac{p_k^T A p_k}{r_k^T r_k} = \frac{r_{k+1}^T r_{k+1} - r_{k+1}^T(\beta_k p_{k-1} - p_k)}{r_k^T r_k} = \frac{r_{k+1}^T r_{k+1}}{r_k^T r_k}.$$

Thus, we derive the simplified Conjugate Gradient (CG) iteration formula from the preliminary Conjugate Gradient (CG) iteration formula.

6. Write a code to implement the Conjugate Gradient (CG) algorithm to solve the quadratic programming (QP) problems:

$$\min_x \frac{1}{2}x^T A x - b^T x,$$

where A is a positive definite 100×100 matrix.
1) A with uniformly distributed eigenvalues.
2) A with 2-3 clusters of eigenvalues.
Output the residual at each iteration and count the number of matrix-vector multiplications.[20pts]

Solution:

```
import numpy as np

# Conjugate Gradient algorithm to solve Ax = b
```

```python
def conjugate_gradient(A, b, tol=1e-6, max_iter=1000):
n = len(b)
x = np.zeros(n)
r = b - A @ x
p = r
residuals = []
mat_vec_count = 0

for k in range(max_iter):
Ap = A @ p
mat_vec_count += 1
alpha = np.dot(r, r) / np.dot(p, Ap)
x += alpha * p
r_new = r - alpha * Ap
residuals.append(np.linalg.norm(r_new))
if np.linalg.norm(r_new) < tol:
break
beta = np.dot(r_new, r_new) / np.dot(r, r)
p = r_new + beta * p
r = r_new
return x, residuals, mat_vec_count

# Generate A with uniformly distributed eigenvalues
def generate_uniform_A(n):
Q = np.random.randn(n, n)
Q, _ = np.linalg.qr(Q)
eigenvalues = np.random.uniform(0.1, 10, size=n)
A = Q @ np.diag(eigenvalues) @ Q.T
return A

# Generate A with clustered eigenvalues
def generate_clustered_A(n):
Q = np.random.randn(n, n)
Q, _ = np.linalg.qr(Q)
eigenvalues = np.concatenate([np.random.normal(1, 0.1, size=n // 3),
np.random.normal(5, 0.2, size=n // 3),
np.random.normal(10, 0.3, size=n-2*(n // 3))])
np.random.shuffle(eigenvalues)
A = Q @ np.diag(eigenvalues) @ Q.T  # A = Q * D * Q.T
return A

# Main function to test the conjugate gradient method
def main():
n = 10
```

```
b = np.random.randn(n)

# Generate A with uniformly distributed eigenvalues
A_uniform = generate_uniform_A(n)

# Solve the quadratic problem using Conjugate Gradient
x_uniform, residuals_uniform, mat_vec_count_uniform = conjugate_gradient(A_uniform, b)

print("Conjugate Gradient with Uniform Eigenvalues:")
print("Residual at each iteration:", residuals_uniform)
print("Number of matrix-vector multiplications:", mat_vec_count_uniform)

# Generate A with clustered eigenvalues
A_clustered = generate_clustered_A(n)

# Solve the quadratic problem using Conjugate Gradient
x_clustered, residuals_clustered, mat_vec_count_clustered = conjugate_gradient(A_clustered, b)

print("\nConjugate Gradient with Clustered Eigenvalues:")
print("Residual at each iteration:", residuals_clustered)
print("Number of matrix-vector multiplications:", mat_vec_count_clustered)

if __name__ == "__main__":
main()
```

1) Conjugate Gradient with Uniform Eigenvalues:

Residual at each iteration:

[5.169894959300269, 3.7392404290561494, 3.0008180033976313, 3.3115065468953437, 2.740471796115691, 1.4108447074842965, 1.005011992456244, 0.6991816683838671, 0.34705354229234114, 0.2847502730561696, 0.22556164640232565, 0.23171901479967222, 0.20662880749497223, 0.16686984008187333, 0.15089594511689072, 0.10875554339542888, 0.09863161580317795, 0.07868770633862061, 0.05251366627512177, 0.03427582510580179, 0.026095163231704355, 0.017591302930239045, 0.014711331224339706, 0.010444712806052036, 0.007026244002107074, 0.005050418974069619, 0.0033742066237358093, 0.0023635926848029586, 0.001795338254547266, 0.0012812722077705297, 0.000865090994102751, 0.0006015387945271139, 0.0003261157886889722, 0.0002536254991547981, 0.0002246865753562579, 0.00012198544482934101, 6.484170838400927e-05, 3.695228005941589e-05, 2.155433496887668e-05, 1.5068116717848343e-05, 1.5147285598976634e-05, 1.3548610319199826e-05, 8.646217963873538e-06, 3.942956851444215e-06, 2.554499067635557e-06, 2.71614907540718e-06, 2.104559326381829e-06, 1.6625889374415396e-06, 6.846126490045413e-07]

Number of matrix-vector multiplications: 49

2) Conjugate Gradient with Clustered Eigenvalues:

Residual at each iteration:

[5.31926729790938, 4.055122899163675, 1.4498736530712544, 0.3369145433830931, 0.2777763669084279, 0.22532911586652246, 0.03192878728378892, 0.02773471200091442, 0.021173392880193476, 0.0034648443932144284, 0.0022876319373354367, 0.0018626350505418267, 0.0004078552205091622, 0.00016359311562878594, 0.0001325561254927638, 6.340304314447973e-05, 1.0728168993587029e-05, 9.461720101462244e-06, 8.43875642295306e-06, 9.478107270508043e-07]

Number of matrix-vector multiplications: 20