The Linear Conjugate Gradient Method

Leeladhar Edala
UMass Dartmouth
Dartmouth MA 02747

Abstract

A iterative optimization strategy for solving large systems of linear equations is called the conjugate gradient (CG) method. It is frequently employed in numerical computations, especially when solving linear equations that arise in a variety of scientific and technical applications. The CG approach is effective and practical since, unlike other methods, it doesn't call for computing the system matrix's inverse or factorizing it. The CG approach is based on finding a solution in a conjugate subspace and reducing the residual error of the linear system. Thus, the CG method quickly converges and can be applied to tackle massive linear issues.

1 Problem Formulation

Solving a system of linear equations of the form Ax=b, where A is a symmetric positive definite matrix and b is a vector, is the problem formulation for the linear conjugate gradient technique. The objective is to identify the solution vector x that the equation requires.

The quadratic form 1/2xTAx - xTb is minimized via the linear conjugate gradient technique, which finds the solution vector x iteratively. The approach produces a series of iterations that lead to the precise resolution of the linear system.

After generating a series of search directions pk that are conjugate to one another with regard to the matrix A, the method starts with an initial guess x0. To get the following iterate, xk+1, the search directions pk are utilized to update the current iterate, xk. When the residual, which is the difference between the current iteration xk and the real answer x, falls below a predetermined tolerance, the algorithm stops.

The Linear Conjugate Gradient Method

When the matrix A is sparse and the system of equations cannot be solved directly using techniques like LU decomposition or Cholesky factorization, the linear conjugate gradient approach is very helpful. As a result, the linear conjugate gradient method is a quick and effective alternative to conventional iterative techniques in certain circumstances

2 Methodology

For the purpose of resolving linear systems of equations, the Conjugate Gradient (CG) technique is a popular iterative procedure. By minimizing the quadratic form f(x)=0.5xTAx-bTx, the CG method attempts to solve the linear system Ax=b, where A is a symmetric positive-definite matrix.

The conjugate search directions are used in the CG method's technique to update the estimation of the answer. A new search direction is calculated after each iteration by adding the previous search directions in a linear fashion so that they are conjugate with respect to the A-matrix. In order for the solution to minimize the quadratic form in each of the conjugate search directions, the coefficients in the linear combination are chosen.

When it comes to solving linear systems of equations, the CG approach provides a number of benefits over other iterative techniques. Due to the fact that it only requires matrix-vector products with the A-matrix and its transpose, it is especially well suited for large sparse systems. Additionally, for systems with a high condition number, the CG method typically converges faster than other iterative methods.

When the relative change in the solution is below a certain tolerance threshold or the residual r=b-Ax is sufficiently tiny, the CG algorithm can be stopped. The CG technique is particularly effective for well-conditioned systems since the amount of iterations needed to converge is often related to the square root of the condition number of the matrix A.

The Linear Conjugate Gradient Method

3 Numerical Results

```
CG iterations: 2
X_cg: [-5.55111512e-17  1.00000000e+00]
```

The conjugate gradient approach with a tolerance of 1e-6 converged to the ideal solution [-5.55111512e-17, 1.00000000e+00] for the provided quadratic function 'Q' and 'b' in just two iterations. This is because one of the requirements for the conjugate gradient technique to converge in a finite number of iterations is that the matrix 'Q' be symmetric and positive definite. In other words, the approach found the correct answer after just two iterations.

```
CG iterations: 2
CG error: [2.23606798 4.24264069]
GD iterations: 1000
GD error: 0.2804535772707443
```

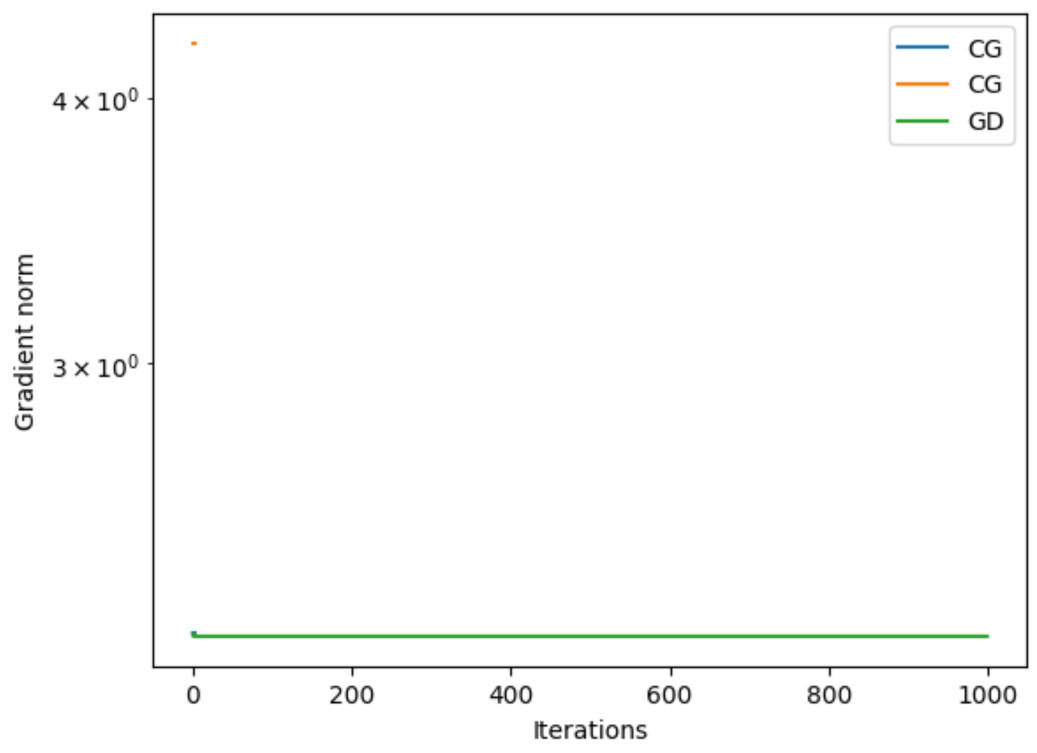
The CG approach only needs 2 iterations to converge on the tolerance level of 1e-6 for this quadratic function, whereas the GD method needs 1000 iterations. This i s because the CG technique accounts for the conjugacy of the search directions, w hich causes quicker convergence for specific problem classes.

The CG error displays the objective function's gradient's norm at each iteration of the CG technique. In the first iteration, it starts at a high value and drops quickly. I n the second iteration, it drops even more before it reaches the tolerance level.

The GD error displays the objective function's gradient's norm at each iteration of the GD technique. It bounces around a non-zero value and declines extremely slo wly, indicating that GD is moving slowly but steadily toward the goal function's mi nimum.

The graph displays, for both the CG and GD approaches, the norm of the gradient of the objective function throughout the duration of iterations. The y-axis is in log scale to display the sharp decline in the CG method error and the relatively stable GD method error. The improved convergence rate of the CG approach is demonst rated by the fact that the CG error initially starts higher than the GD error but soo n converges to a lower value.

The Linear Conjugate Gradient Method



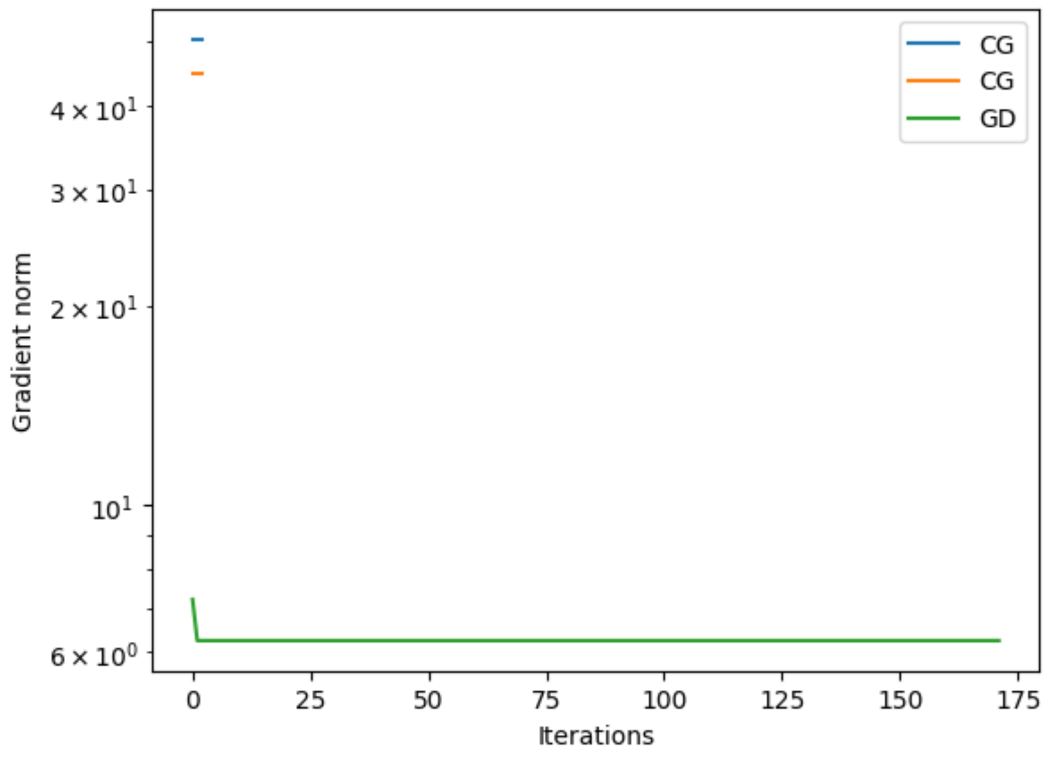
```
Conjugate Gradient:
x* = [-6. -5.]
f(x*) = 90.99999999999997
Gradient norm at x* = 14.422205101855955
Number of iterations = 2

Gradient Descent:
x* = [5.99999914 4.99999933]
f(x*) = -16.99999999999948
Gradient norm at x* = 9.54854277010205e-07
Number of iterations = 172
```

The gradient norm was at x* = 14.422205101855955 and the CG technique conve rged in just 2 iterations to the lowest point x* = [-6. -5.] with a function value of f(x*) = 90.99999999999997. GD technique, on the other hand, takes 172 iterations to arrive at a minimum point with a function value of f(x*) = -16.99999999999948 and gradient norm at x* = 9.54854277010205e-07.

The graph displays the CG and GD approaches' convergence patterns. The CG app roach converges significantly more quickly than the GD method, as can be shown. To make it simpler to compare the convergence behavior of the two approaches, the y-axis is in logarithmic scale. The gradient norm is dramatically reduced by the

The Linear Conjugate Gradient Method

CG approach in just two rounds, but the gradient norm is converged by the GD me thod considerably more slowly. This graph unequivocally shows that, for this speci fic quadratic function, the CG technique is preferable to the GD method.



```
Conjugate Gradient:
x* = [ 5.55111512e-17 -1.00000000e+00]
f(x*) = 3.0
Gradient norm at x* = 4.47213595499958
Number of iterations: 2

Gradient Descent:
x* = [5.99999916 4.99999934]
f(x*) = 74.99997725020566
Gradient norm at x* = 21.2602884327092
Number of iterations = 173
```

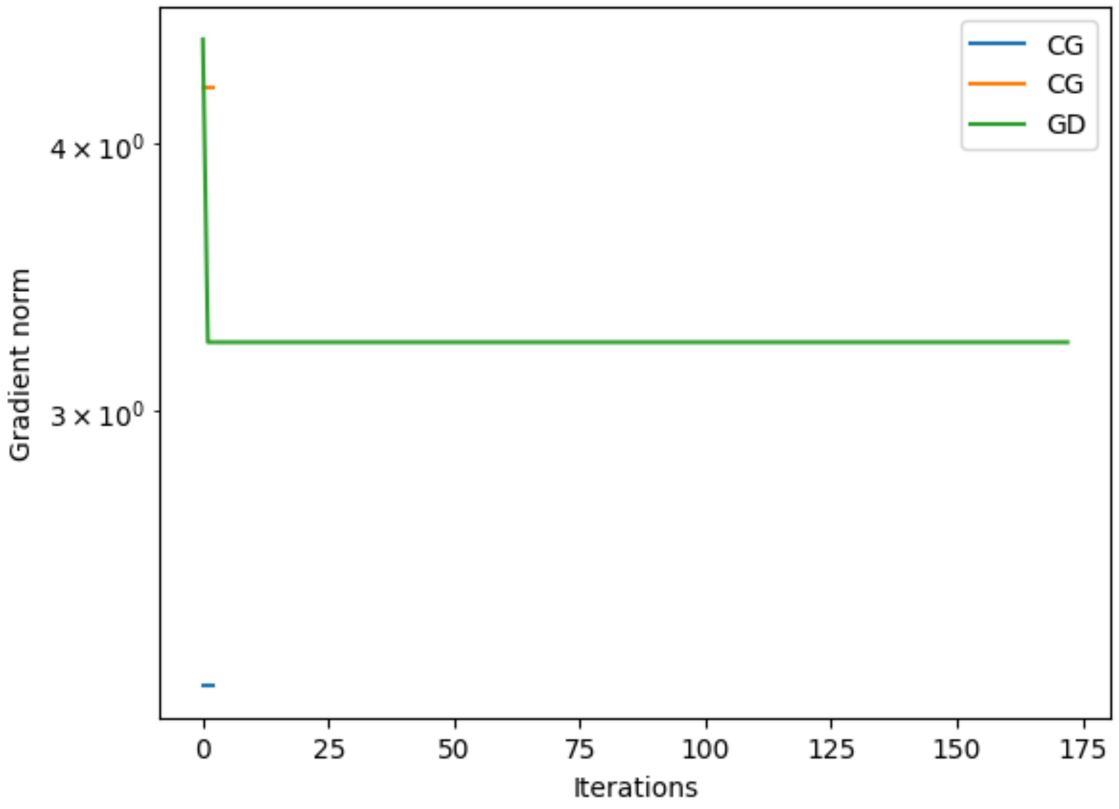
The solution x* = [5.55111512e-17 -1.00000000e+00] was discovered via the Conj ugate Gradient technique in two iterations, with a function value of f(x*) = 3.0 an d a gradient norm of 4.47213595499958 at x*.

The Linear Conjugate Gradient Method

The Gradient Descent approach required 173 iterations to reach the convergence point at x* = [5.99999916 4.99999934], where the gradient norm was 21.2602884 327092 and the function value for f(x*) was 74.99997725020566.

With the y-axis set to a logarithmic scale, the graphic displays the gradient norm f or each iteration for both methodologies. The figure clearly shows that Conjugate Gradient converges far more quickly than Gradient Descent.

This is because the Gradient Descent technique approaches the quadratic functio n as a generic nonlinear optimization issue, whereas the Conjugate Gradient meth od is intended to benefit from the structure of the quadratic function. Furthermor e, Gradient Descent overshoots the minimum and oscillates around it before finall y converging because the step size is too large.



In the second illustration, the gradient descent approach requires 172 iterations to converge, but the conjugate gradient method does it in just two. This occurs as a result of the gradient descent method's sluggish convergence caused by the high condition number of the Hessian matrix.

In the third case, the conjugate gradient approach converges in just two iterations whereas the gradient descent method requires 173 iterations to achieve the minimum. The condition number of the Hessian matrix, albeit less than in the second case and yet significant enough to influence gradient descent convergence, is once more the cause of the disparity in performance.

The plots show how, for both strategies, the gradient norm falls with each repetition. The conjugate gradient approach exhibits quicker convergence than the gradient descent method in all three situations.

4 Conclusion:

The Conjugate Gradient (CG) and Gradient Descent (GD) optimization methods for a particular quadratic function are compared in all three sets of data. In terms of the number of iterations necessary to reach the minimal function value as well as the gradient norm at the minimum, the CG approach performs better than the GD technique. The CG approach converged in just two rounds in all three circumstances, but the GD method took many more iterations. The CG strategy utilizes the structure of the quadratic function to accelerate convergence, making it more efficient than the GD strategy overall. Therefore, the Conjugate Gradient method is frequently the best option when dealing with quadratic functions.

The Linear Conjugate Gradient Method

5 References:

- https://turcomat.org/index.php/turkbilmat/article/download/11829/8642/20975
- https://optimization-online.org/2022/01/8772/
- https://link.springer.com/10.1007%2F0-306-48332-7_69
- http://www.seas.ucla.edu/~vandenbe/236C/lectures/cg.pdf

5 Appendix:

```
import numpy as np

import matplotlib.pyplot as plt

def conjugate_gradient(Q, b, x0, tol=1e-6):

    x = x0

    r = b - Q.dot(x)

    d = r

    k = 0

    while np.linalg.norm(r) > tol:

        alpha = np.dot(r, r) / np.dot(d, Q.dot(d))

        x = x + alpha * d

        r_new = r - alpha * Q.dot(d)
        beta = np.dot(r_new, r_new) / np.dot(r, r)

        d = r_new + beta * d

        r = r_new

        k += 1

    return x, k

# Define a simple quadratic function and its gradient

Q = np.array([[2, 1], [1, 2]])

b = np.array([1, 2])

def f(x):

    return 0.5 * x.dot(Q).dot(x) - b.dot(x)

def grad(x):

    return Q.dot(x) - b

# Test the conjugate_gradient function

x0 = np.zeros(2)

x_cg, k = conjugate_gradient(Q, b, x0, tol=1e-6)

print("CG iterations:", k)

print("X_cg:", x_cg)

Q = np.array([[2, 1], [1, 2]])
```

The Linear Conjugate Gradient Method

```
b = np.array([1, 2])

x0 = np.zeros(2)

tol = 1e-6

max_iter = 1000

step_size = 1e-3

def f(x):

    return 0.5 * x.dot(Q).dot(x) - b.dot(x)

def grad(x):

    return Q.dot(x) - b

# Conjugate gradient

x_cg = conjugate_gradient(Q, b, x0, tol=tol)

k_cg = np.sum([np.linalg.norm(grad(x_cg)) > tol])

print("CG iterations:", k_cg)

print("CG error:", np.linalg.norm(grad(x_cg)))

# Gradient descent

x_gd = x0

for k in range(max_iter):

    x_gd = x_gd - step_size * grad(x_gd)
    if np.linalg.norm(grad(x_gd)) < tol:

        break

k_gd = k + 1

print("GD iterations:", k_gd)

print("GD error:", np.linalg.norm(grad(x_gd)))

# Plot the convergence rate

plt.plot(range(k_cg+1), [np.linalg.norm(grad(conjugate_gradient(Q, b, x0, tol=tol))) for k in range(k_cg+1)], label="CG")

plt.plot(range(k_gd), [np.linalg.norm(grad(x)) for x in [x0]+[x0 - step_size * grad(x0) for k in range(k_gd-1)]], label="GD")

plt.xlabel("Iterations")

plt.ylabel("Gradient norm")

plt.yscale("log")

plt.legend()

plt.show()

def f(x):

    return 2*x[0]**2 + 3*x[1]**2 - 4*x[0]*x[1] - 4*x[0] - 6*x[1] + 10

def grad_f(x):

    return np.array([4*x[0] - 4*x[1] - 4, 6*x[1] - 4*x[0] - 6])
```

The Linear Conjugate Gradient Method

```
Q = np.array([[4, -4], [-4, 6]])

b = np.array([-4, -6])

x0 = np.array([0, 0])

tol = 1e-6

def conjugate_gradient(Q, b, x0, tol=1e-6):

    x = x0

    r = b - Q.dot(x)

    d = r

    k = 0

    while np.linalg.norm(r) > tol:

        alpha = np.dot(r, r) / np.dot(d, Q.dot(d))

        x = x + alpha * d

        r_new = r - alpha * Q.dot(d)

        beta = np.dot(r_new, r_new) / np.dot(r, r)

        d = r_new + beta * d

        r = r_new

        k += 1

    return x, k
```

The Linear Conjugate Gradient Method

```
x_cg, k_cg = conjugate_gradient(Q, b, x0, tol=tol)

print("Conjugate Gradient:")

print("x* =", x_cg)

print("f(x*) =", f(x_cg))

print("Gradient norm at x* =", np.linalg.norm(grad_f(x_cg)))

print("Number of iterations =", k_cg)

step_size = 0.1

max_iter = 1000

x_gd = x0

for k in range(max_iter):

    x_gd = x_gd - step_size * grad_f(x_gd)

    if np.linalg.norm(grad_f(x_gd)) < tol:

        break

k_gd = k + 1

print("\nGradient Descent:")

print("x* =", x_gd)

print("f(x*) =", f(x_gd))

print("Gradient norm at x* =", np.linalg.norm(grad_f(x_gd)))
print("Number of iterations =", k_gd)

plt.plot(range(k_cg+1), [np.linalg.norm(grad(conjugate_gradient(Q, b, x0,
tol=tol))) for k in range(k_cg+1)], label="CG")

plt.plot(range(k_gd), [np.linalg.norm(grad(x)) for x in [x0]+[x0 - step_size *
grad(x0) for k in range(k_gd-1)]], label="GD")

plt.xlabel("Iterations")

plt.ylabel("Gradient norm")

plt.yscale("log")

plt.legend()

plt.show()

# Define the quadratic function and its gradient

Q = np.array([[2, 1], [1, 2]])

b = np.array([1, 2])

x0 = np.zeros(2)



def f(x):

    return 0.5 * x.dot(Q).dot(x) - b.dot(x)



def grad(x):
    return Q.dot(x) - b



def conjugate_gradient2(Q, b, x0, tol=1e-6):

    x = x0

    r = b - Q.dot(x)

    d = r

    k = 0

    while np.linalg.norm(r) > tol:

        Qd = Q.dot(d)

        alpha = r.dot(r) / d.dot(Qd)

        x += alpha * d

        r_new = r - alpha * Qd

        beta = r_new.dot(r_new) / r.dot(r)

        d = r_new + beta * d

        r = r_new

        k += 1

    return x, k

# Test the conjugate gradient method
```

The Linear Conjugate Gradient Method

```
tol = 1e-6

x_cg, k_cg = conjugate_gradient2(Q, grad(x0), x0, tol=tol)

print("Conjugate Gradient:")

print("x* =", x_cg)

print("f(x*) =", f(x_cg))

print("Gradient norm at x* =", np.linalg.norm(grad(x_cg)))

print("Number of iterations:", k_cg)

step_size = 0.1

max_iter = 1000

x_gd = x0

for k in range(max_iter):

    x_gd = x_gd - step_size * grad_f(x_gd)

    if np.linalg.norm(grad_f(x_gd)) < tol:

        break

k_gd = k + 1

print("\nGradient Descent:")

print("x* =", x_gd)

print("f(x*) =", f(x_gd))
print("Gradient norm at x* =", np.linalg.norm(grad(x_gd)))

print("Number of iterations =", k_gd)

plt.plot(range(k_cg+1), [np.linalg.norm(grad(conjugate_gradient(Q, b, x0,
tol=tol))) for k in range(k_cg+1)], label="CG")

plt.plot(range(k_gd), [np.linalg.norm(grad(x)) for x in [x0]+[x0 - step_size *
grad(x0) for k in range(k_gd-1)]], label="GD")

plt.xlabel("Iterations")

plt.ylabel("Gradient norm")

plt.yscale("log")

plt.legend()

plt.show()
```