非线性方程求解

1. 问题描述

考虑以下非线性方程组：

$$
\begin{aligned}
&3x_1 - \cos(x_2 x_3) = 0.5 \\
&x_1^2 - 81(x_2 + 0.1)^2 + \sin(x_3) = -1.06 \\
&\exp(-x_1 x_2) + 20x_3 + \frac{10\pi - 3}{3} = 0
\end{aligned}
$$

设初始值为 $x^{(0)} = (0.1, 0.1, -0.1)^T$。
要求用牛顿迭代法、简化的牛顿迭代法和割线迭代法，求解向量 $x = \{x_1, x_2, x_3\}^T$，要求精度 $\|x^{(k)} - x^{(k-1)}\|_\infty$ 不低于 10^{-6} (即 $\|x^{(k)} - x^{(k-1)}\|_\infty < 10^{-6}$ 时停止迭代)。

2. 求解方法

2.1 牛顿迭代法

牛顿迭代法通过泰勒级数展开，利用函数在当前点的值及其导数信息（雅可比矩阵），来迭代求解非线性方程组的根。对于给定的初始猜测 $x^{(0)}$，迭代公式为：

$$
x^{(k+1)} = x^{(k)} - J\big(x^{(k)}\big)^{-1} F\big(x^{(k)}\big)
$$

其中，$J\big(x^{(k)}\big)$ 为 $F\big(x\big)$ 在 $x^{(k)}$ 处的雅可比矩阵。

2.2 简化牛顿迭代法

简化牛顿迭代法通过在整个迭代过程中保持雅可比矩阵的某种形式不变（如初始时刻的雅可比矩阵），来减少每次迭代中对雅可比矩阵的重新计算。具体而言，假设在 $x^{(0)}$ 处的雅可比矩阵为 $J\big(x^{(0)}\big)$，则迭代公式为：

$$
x^{(k+1)} = x^{(k)} - J\big(x^{(0)}\big)^{-1} F\big(x^{(k)}\big)
$$

2.3 割线法

割线法是一种拟牛顿法，通过对雅可比矩阵的低秩更新，逐步逼近真实的雅可比矩阵。其基本思想是用一个简单的矩阵 B_k 来近似真实的雅可比矩阵 $J(x)$，并通过以下迭代公式更新 B_k：

$$x^{(k+1)} = x^{(k)} - B_k^{-1} F(x^{(k)})$$

更新公式为：

$$B\_k + 1 = B_k + \frac{(y_k - B_k s_k) s_k^T}{s_k^T s_k}$$

其中，$s_k = x^{(k+1)} - x^{(k)}$，$y_k = F(x^{(k+1)}) - F(x^{(k)})$。

3. 实验结果

使用 MATLAB 编程实现上述三种方法。初始猜测值 $x^{(0)} = (0.1, 0.1, -0.1)^T$，收敛判据为 $\|x^{(k)} - x^{(k-1)}\|_\infty < 10^{-6}$，最大迭代次数设置为 100。

计算结果如下表所示：

方法	收敛解 x^* (x_1, x_2, x_3)	迭代次数	最终误差 $\|x^{(k)} - x^{(k-1)}\|_\infty$
牛顿法	(0.50000000, 0.00000000, -0.52359878)	5	7.757857e-10
简化牛顿法	(0.50000000, 0.00000100, -0.52359873)	15	9.947985e-07
割线法	(0.50000000, 0.00000000, -0.52359878)	6	1.935434e-07

参考近似解为 $x_{approx} = (0.5, 0, -0.52359877)^T$。
各方法得到的解与参考近似解的无穷范数误差：

- 牛顿法: $\|x^*_{Newton} - x_{approx}\|_\infty \approx 5.60 \times 10^{-09}$
- 简化牛顿法: $\|x^*_{Simplified} - x_{approx}\|_\infty \approx 9.97 \times 10^{-07}$
- 割线法: $\|x^*_{Broyden} - x_{approx}\|_\infty \approx 5.60 \times 10^{-09}$

4. 结论

三种方法均能收敛到问题的近似解。

- **牛顿法** 收敛速度最快（迭代 5 次），且最终解的精度较高，与参考近似解的误差较小。但每步迭代都需要计算雅可比矩阵并求解线性方程组，计算量相对较大。
- **简化牛顿法** 通过固定雅可比矩阵减少了每步的计算量，但收敛速度最慢（迭代 15 次），其解与参考近似解的误差也相对较大，但仍在精度要求之内。
- **割线法** 作为一种拟牛顿法，在计算效率和收敛速度之间取得了较好的平衡，其迭代次数（6 次）少于简化牛顿法，且最终解的精度与牛顿法在当前参考解的比较下相近。

对于本问题，牛顿法和割线法在解的精度上表现相似且优于简化牛顿法。牛顿法迭代次数最少。所有方法均在设定的最大迭代次数内达到了要求的精度。

5. 源代码

```
clear
clc;
format long;

% 定义非线性方程组 F(x)
F_func = @(x) [
    3*x(1) - cos(x(2)*x(3)) - 0.5;
    x(1)^2 - 81*(x(2) + 0.1)^2 + sin(x(3)) + 1.06;
    exp(-x(1)*x(2)) + 20*x(3) + (10*pi - 3)/3
];

% 定义雅可比矩阵 J(x)
J_func = @(x) [
    3,            x(3)*sin(x(2)*x(3)),      x(2)*sin(x(2)*x(3));
    2*x(1),       -162*(x(2)+0.1),          cos(x(3));
    -x(2)*exp(-x(1)*x(2)), -x(1)*exp(-x(1)*x(2)), 20
];

% 初始参数
x_initial = [0.1; 0.1; -0.1]; % 初始猜测值
tol = 1e-6;                   % 精度要求
max_iter = 100;               % 最大迭代次数

fprintf('初始猜测值 x0 = [%.1f, %.1f, %.1f]''\n', x_initial(1), x_initial(2), x_initial(3));
fprintf('精度要求 ||x(k) - x(k-1)||_inf < %e\n\n', tol);

% --- 牛顿迭代法 ---
fprintf('牛顿迭代法:\n');
x_current = x_initial;
iter_count = 0;
err_norm = inf; % 初始化误差范数

for k = 1:max_iter
    Fx = F_func(x_current);
    Jx = J_func(x_current);

    % 检查雅可比矩阵是否奇异
    if rank(Jx) < length(x_initial)
        fprintf('雅可比矩阵在第 %d 次迭代时奇异。牛顿法失败。\n', k);
        x_current = nan(size(x_initial)); % 标记失败
```

```
        iter_count = k;
        break;
    end

    delta_x = Jx \ (-Fx); % 解线性方程组 Jx * delta_x = -Fx
    x_next = x_current + delta_x;
    iter_count = k;
    err_norm = norm(delta_x, inf); % 计算误差 ||x_next - x_current||_inf

    x_current = x_next;

    if err_norm < tol
        break; % 达到精度要求
    end

    if k == max_iter
        fprintf('牛顿法在 %d 次迭代内未收敛。\n', max_iter);
    end
end
results_newton.x = x_current;
results_newton.iter = iter_count;
results_newton.err = err_norm;

fprintf('解: [%.8f, %.8f, %.8f]''\n', results_newton.x(1), results_newton.x(2), results_newton.x
fprintf('迭代次数: %d\n', results_newton.iter);
fprintf('最终误差 ||x(k) - x(k-1)||_inf: %e\n\n', results_newton.err);


% --- 简化牛顿迭代法 ---
fprintf('简化牛顿迭代法:\n');
x_current = x_initial;
iter_count = 0;
err_norm = inf;
J0 = J_func(x_initial); % 计算一次雅可比矩阵 J(x0)

% 检查初始雅可比矩阵是否奇异
if rank(J0) < length(x_initial)
    fprintf('初始雅可比矩阵 J0 奇异。简化牛顿法无法启动。\n');
    results_simp_newton.x = nan(size(x_initial));
    results_simp_newton.iter = 0;
    results_simp_newton.err = inf;
else
    for k = 1:max_iter
```

```
        Fx = F_func(x_current);
        delta_x = J0 \ (-Fx); % 使用 J0 求解
        x_next = x_current + delta_x;
        iter_count = k;
        err_norm = norm(delta_x, inf);

        x_current = x_next;

        if err_norm < tol
            break;
        end

        if k == max_iter
            fprintf('简化牛顿法在 %d 次迭代内未收敛。\n', max_iter);
        end
    end
    results_simp_newton.x = x_current;
    results_simp_newton.iter = iter_count;
    results_simp_newton.err = err_norm;
end

fprintf('解: [%.8f, %.8f, %.8f]''\n', results_simp_newton.x(1), results_simp_newton.x(2), result
fprintf('迭代次数: %d\n', results_simp_newton.iter);
fprintf('最终误差 ||x(k) - x(k-1)||_inf: %e\n\n', results_simp_newton.err);


% --- 割线法 (Broyden法) ---
fprintf('割线法 (Broyden法):\n');
x_current = x_initial;
iter_count = 0;
err_norm = inf;

B_k = J_func(x_initial);         % 初始化 B0 = J(x0)
Fx_current_val = F_func(x_current); % F(x0)

for k = 1:max_iter
    % 检查 Bk 是否奇异
    if rank(B_k) < length(x_initial)
        fprintf('矩阵 B_k 在第 %d 次迭代时奇异。Broyden法失败。\n', k);
        x_current = nan(size(x_initial));
        iter_count = k;
        break;
    end
```

```
    delta_x = B_k \ (-Fx_current_val); % 求解 B_k * delta_x = -F(x_k)
    x_next = x_current + delta_x;
    iter_count = k;
    err_norm = norm(delta_x, inf);

    if err_norm < tol
        x_current = x_next; % 更新解后退出
        break;
    end

    Fx_next_val = F_func(x_next); % F(x_{k+1})

    s_k = x_next - x_current; % s_k = delta_x
    y_k = Fx_next_val - Fx_current_val; % y_k = F(x_{k+1}) - F(x_k)

    % 更新 B_k -> B_{k+1}
    sksk_inner_product = s_k' * s_k;
    if abs(sksk_inner_product) < eps % 避免除以过小的数
        fprintf('分母 s_k''*s_k (%.2e) 在第 %d 次迭代时过小。Broyden法更新可能不稳定或失败。\n', sl
        x_current = x_next; % 保留当前计算的解
        break;
    end
    B_k = B_k + (y_k - B_k * s_k) * s_k' / sksk_inner_product;

    x_current = x_next;
    Fx_current_val = Fx_next_val;

    if k == max_iter
        fprintf('Broyden法在 %d 次迭代内未收敛。\n', max_iter);
    end
end
results_broyden.x = x_current;
results_broyden.iter = iter_count;
results_broyden.err = err_norm;

fprintf('解: [%.8f, %.8f, %.8f]''\n', results_broyden.x(1), results_broyden.x(2), results_broyde
fprintf('迭代次数: %d\n', results_broyden.iter);
fprintf('最终误差 ||x(k) - x(k-1)||_inf: %e\n\n', results_broyden.err);


% 近似解用于比较 (0.5, 0, -0.52359877)
approx_sol = [0.5; 0; -0.52359877];
```

```
fprintf('\n参考近似解: [%.8f, %.8f, %.8f]''\n', approx_sol(1), approx_sol(2), approx_sol(3));
fprintf('牛顿法与近似解的差的范数: %e\n', norm(results_newton.x - approx_sol, inf));
fprintf('简化牛顿法与近似解的差的范数: %e\n', norm(results_simp_newton.x - approx_sol, inf));
fprintf('割线法与近似解的差的范数: %e\n', norm(results_broyden.x - approx_sol, inf));
```