2021 Spring MAS 365: Homework 5

posted on Apr 8; due by Apr 15

1. [10+10 points] The nonlinear system

$$x_1^2 - 10x_1 + x_2^2 + 8 = 0, \quad x_1x_2^2 + x_1 - 10x_2 + 8 = 0$$

can be transformed into the fixed-point problem

$$x_1 = g_1(x_1, x_2) = \frac{x_1^2 + x_2^2 + 8}{10}, \quad x_2 = g_2(x_1, x_2) = \frac{x_1x_2^2 + x_1 + 8}{10}.$$

(a) Use Theorem 10.6 in the textbook to show that $G = (g_1, g_2)^t$ mapping $D \subset \mathbb{R}^2$ to $\mathbb{R}^2$ has a unique fixed point in

$$D = \{(x_1, x_2)^t \mid 0 \le x_1, x_2 \le 1.5\}.$$

(b) Use Theorem 10.6 to estimate the number of iterations required for the fixed-point iteration to achieve 10^{-3} accuracy of $||\boldsymbol{x}^{(k)} - \boldsymbol{p}||_\infty$ with $\boldsymbol{x}^{(0)} = (0,0)^t$, where $\boldsymbol{p}$ is the unique fixed point.

Solution:

(a) G is a polynomial, so it is continuous. Since

$$0.8 \le \frac{x_1^2 + x_2^2 + 8}{10} \le 1.25, \quad 0.8 \le \frac{x_1x_2^2 + x_1 + 8}{10} \le 1.2875$$

for all $\boldsymbol{x} \in D$, we have that $G(\boldsymbol{x}) \in D$ whenever $x \in D$. In addition, since

$$\left|\frac{\partial g_1(\boldsymbol{x})}{\partial x_1}\right| = \left|\frac{2x_1}{10}\right| \le \frac{3}{10}, \quad \left|\frac{\partial g_1(\boldsymbol{x})}{\partial x_2}\right| = \left|\frac{2x_2}{10}\right| \le \frac{3}{10},$$

$$\left|\frac{\partial g_2(\boldsymbol{x})}{\partial x_1}\right| = \left|\frac{x_2^2 + 1}{10}\right| \le \frac{3.25}{10}, \quad \left|\frac{\partial g_1(\boldsymbol{x})}{\partial x_2}\right| = \left|\frac{2x_1x_2}{10}\right| \le \frac{4.5}{10},$$

we have that

$$\left|\frac{\partial g_i(\boldsymbol{x})}{\partial x_j}\right| \le 0.45 = \frac{0.9}{2}$$

for all $i, j = 1, 2$. Therefore, by Theorem 10.6, G has a unique fixed point in D.

(b) We find an integer k that satisfies

$$||\boldsymbol{x}^{(k)} - \boldsymbol{p}||_\infty \le \frac{K^k}{1-K}||\boldsymbol{x}^{(1)} - \boldsymbol{x}^{(0)}||_\infty < 10^{-3},$$

where $K = 0.9$ and $\boldsymbol{x}^{(1)} = (0.8, 0.8)^t$. This reduces to

$$\frac{0.9^k}{0.1}0.8 < 10^{-3}$$

which is equivalent to

$$k \log_{10} 0.9 < -3 - \log_{10} 8 \quad \text{and} \quad k > \frac{-3 - \log_{10} 8}{\log_{10} 0.9} \approx 85.2995$$

Hence, 86 iterations will ensure the desired accuracy.

2. [10 points] What does Newton's method reduce to for the linear system $A\boldsymbol{x} = \boldsymbol{b}$ given by

$$\begin{aligned} a_{11}x_1 + a_{12}x_2 + \cdots + a_{1n}x_n &= b_1, \\ a_{21}x_1 + a_{22}x_2 + \cdots + a_{2n}x_n &= b_2, \\ &\vdots \\ a_{n1}x_1 + a_{n2}x_2 + \cdots + a_{nn}x_n &= b_n, \end{aligned}$$

where A is a nonsingular matrix?

Solution: Let $f_j(x_1, \ldots, x_n) = a_{j1}x_1 + \cdots + a_{jn}x_n - b_j$. Since $\frac{\partial f_j}{\partial x_i} = a_{ji}$, the Jacobian matrix is $J(\boldsymbol{x}) = A$. Then, one iteration of Newton's method is as follows:

$$\boldsymbol{x}^{(1)} = \boldsymbol{x}^{(0)} - J(\boldsymbol{x}^{(0)})^{-1}F(\boldsymbol{x}^{(0)}) = \boldsymbol{x}^{(0)} - A^{-1}(A\boldsymbol{x}^{(0)} - \boldsymbol{b}) = A^{-1}\underline{.}$$

Therefore, for any given $\boldsymbol{x}^{(0)}$, one iteration of Newton's method finds the solution.

3. [10+10+10 points]

(a) Implement the conjugate gradient method via MATLAB grader.

(b) Implement the preconditioned conjugate gradient method via MATLAB grader.

(c) Implement Newton's method via MATLAB grader.

Solution:

(a)
```
function [xc Nc] = conjugate_gradient(A, b, x0, epsilon, N)
    x = x0;
    r = b - A*x;
    v = r;
    rr0 = r'*r;
    for k=1:N
        xprev = x;
        Av = A*v;
        t = rr0/(v'*Av);
        x = x + t*v;
        r = r - t*Av;
        rr1 = r'*r;
        s = rr1/rr0;
        v = r + s*v;
        rr0 = rr1;
        if (norm(x - xprev, 2)/norm(x,2) < epsilon)
            break;
        end
    end
    xc = x;
    Nc = k;
end
```

(b)
```
function [xp Np] = preconditioned_conjugate_gradient(A, b, x0, epsilon, N)
    D = diag(diag(A));
    sqrt_invD = sqrt(inv(D));
    Ap = sqrt_invD*A*sqrt_invD;
    bp = sqrt_invD*b;
```

```
        x = sqrt(D)*x0;
        r = bp - Ap*x;
        v = r;
        rr0 = r'*r;
        for k=1:N
            xprev = x;
            Av = Ap*v;
            t = rr0/(v'*Av);
            x = x + t*v;
            r = r - t*Av;
            rr1 = r'*r;
            s = rr1/rr0;
            v = r + s*v;
            rr0 = rr1;
            if (norm(sqrt_invD*(x - xprev), 2)/norm(sqrt_invD*x,2) < epsilon)
                break;
            end
        end
        xp = sqrt_invD*x;
        Np = k;
    end
```

(c)
```
    function sol = newton(x0, N, epsilon)
        x = x0;
        for k=1:N
            F = [3*x(1) - cos(x(2)*x(3)) - 1/2;
                 4*x(1)^2 - 625*x(2)^2 + 2*x(2) - 1;
                 exp(-x(1)*x(2)) + 20*x(3) + (10*pi-3)/3];
            J = [3 x(3)*sin(x(2)*x(3)) x(2)*sin(x(2)*x(3));
                 8*x(1) -1250*x(2)+2 0;
                 -x(2)*exp(-x(1)*x(2)) -x(1)*exp(-x(1)*x(2)) 20];
            x_prev = x;
            x = x - J\F;
            if (norm(x - x_prev, Inf)/norm(x, Inf) < epsilon)
                break;
            end
        end
        sol = [x; k];
    end
```