Problem Sheet 5

Numerical Methods for CSE

David Bimmler

1. Modified Newton Method

1.a. Consistency of Iteration

In order to prove consistency of the given iteration function it suffices to prove $x^{(0)} = x^{(k)}$ for $k \in \mathbb{N}$.

Proof. We proceed by induction. The following facts are given:

$$\mathbf{F}(x^{(0)}) = 0 \text{ and } D\mathbf{F}(x^{(0)})^{-1} \text{ regular} \tag{1}$$

The induction hypothesis is as follows:

$$P(n) \equiv x^{(0)} = x^{(n)}$$

Trivially, the base case $n = 0$ holds. We finish the proof by showing that $P(n+1)$ holds, given $P(n)$ for $n \in \mathbb{N}$.

$$x^{(n+1)} = y^{(n)} - D\mathbf{F}(x^{(n)})^{-1}\mathbf{F}(y^{(n)}) = x^{(0)} - D\mathbf{F}(x^{(0)})^{-1}\mathbf{F}(x^{(0)}) \overset{(1)}{=} x^{(0)}$$

since

$$\begin{aligned} y^{(n)} &= x^{(n)} + D\mathbf{F}(x^{(n)})^{-1}\mathbf{F}(x^{(n)}) \\ &\overset{\text{I.H.}}{=} x^{(0)} + D\mathbf{F}(x^{(0)})^{-1}\mathbf{F}(x^{(0)}) \\ &\overset{(1)}{=} x^{(0)} \end{aligned}$$

1.b. Eigen implementation of `mod_newt_step`

```
#include <iostream>
#include <Eigen/Dense>
#include <vector>

using namespace std;
```

```
template <typename arg, class func, class jac>
void mod_newt_step(const arg & x, arg & x_next, func&& f, jac&& df) {
    arg y = x + (1 / df(x)) * f(x);
    x_next = y - (1 / df(x)) * f(y);
};
```

1.c. Eigen implementation of `mod_newt_ord`

Not sure whether the order of convergence calculation works…

```
void mod_newt_ord() {
    double x, x_next, p;
    const double x_star = 0.123624065869274, tol = 2e-16;
    int max_iter = 100;
    double err[3] = {0,0,0};
    vector<double> ps;

    auto f = [](double d){
        return atan(d) - 0.123;
    };
    auto df = [](double d){
        return (1/(pow(d, 2) + 1));
    };
    x_next = 5; // Initial guess

    for (int i = 0; i < max_iter; i++) {
        err[i % 3] = x_next;

        if (i > 2) {
            p = (log(err[2]) - log(err[1]))/(log(err[1]) / log(err[0]));
            ps.push_back(p);
        }

        x = x_next;
        mod_newt_step(x, x_next, f, df);

        if (abs(x - x_next) < tol || abs(x - x_next) < abs(x_next) * tol) {
            cout << "Abort condition reached" << endl;
            break;
        }
    }
    cout << "x_next: " << x_next << endl;
    for (vector<double>::iterator it = ps.begin(); it != ps.end(); ++it) {
        cout << "p: " << *it << endl;
    }
}
```