Lab problem 9/7/2005

Problem. In most solids, the internal stresses (forces) σ increase linearly with the displacement x if you stretch them, i.e. $\sigma = Ax$. It is therefore relatively simple to determine the maximum displacement x^* at which the break stress σ^* is exceeded: if $x > x^* = \sigma^*/A$, then the body breaks.

However, this linear relationship does not hold for plastic materials like rubber. Assume that for this material, $\sigma = \sqrt{x + 0.2} + \frac{x}{10} - \sqrt{0.2}$. Determine the maximum deflection if the breaking stress is $\sigma^* = 1$ using Newton's method.

Solution. We need to find that value of x for which $\sqrt{x + 0.2} + \frac{x}{10} - \sqrt{0.2} = \sigma^* = 1$, i.e. we need to find a zero for the function

$$f(x) = \sqrt{x + 0.2} + \frac{x}{10} - \sqrt{0.2} - 1.$$

If we plot this function, we see that the zero lies somewhere in the range between $x = 1$ and $x = 2$, so we start a Newton iteration at $x_0 = 1$. The iteration formula then reads

$$\begin{aligned} x_{k+1} &= x_k - \frac{f(x_k)}{f'(x_k)} \\ &= x_k - \frac{\sqrt{x_k + 0.2} + \frac{x}{10} - \sqrt{0.2} - 1}{\frac{1}{2\sqrt{x+0.2}} + \frac{1}{10}}. \end{aligned}$$

We implement this in the following program:

```
#include <iostream>
#include <iomanip>
#include <cmath>

// define f(x)
double f(double x)
{
  return std::sqrt(x+0.2) + x/10 - std::sqrt(0.2) - 1;
}


// also define f'(x)
double f_prime (double x)
{
  return 1./(2*std::sqrt(x+0.2)) + 1./10;
}


int main ()
{
```

```
                                    // set output precision to all 16 valid
                                    // digits of double precision floating
                                    // point numbers
  std::cout << std::setprecision(16);

                                    // now define the iteration; start at
                                    // x_0=1
  double x = 1;
  std::cout << "x_0 = " << x << std::endl;

                                    // do 10 iterations and output the result
  for (int i=1; i<=10; ++i)
    {
      x = x - f(x)/f_prime(x);
      std::cout << "x_" << i << " = " << x << std::endl;
    }
}
```

Running this program yields the following output:

$$
\begin{aligned}
x_0 &= 1 \\
x_1 &= 1.452466631824421 \\
x_2 &= 1.486178871522092 \\
x_3 &= 1.48631536197905 \\
x_4 &= 1.486315364171695 \\
x_5 &= 1.486315364171695 \\
x_6 &= 1.486315364171695 \\
x_7 &= 1.486315364171695 \\
x_8 &= 1.486315364171695 \\
x_9 &= 1.486315364171695 \\
x_{10} &= 1.486315364171695
\end{aligned}
$$

If we accept the last number as exact up to machine precision, then x_1 has 2 correct digits, x_2 has 4, x_3 has 9, and x_4 already all digits correct. This corresponds with theory that predicts that the number of correct digits doubles in each iteration if the constant C in the convergence formula $e_{k+1} = Ce_k^2$ is approximately or less than 1. Indeed, let us use the definition

$$
C = \frac{1}{2}\frac{|f''(\xi)|}{|f'(x^*)|} = \frac{1}{2}\frac{\frac{1}{4\sqrt{(\xi+0.2)^3}}}{\frac{1}{2\sqrt{x^*+0.2}} + \frac{1}{10}}
$$

for some point ξ in the vicinity of the starting point x_0 and the solution x^*. We

know that $x^* \approx 1.5$, and use $\xi = 1.5$ to get an approximation for C:

$$C \approx \frac{1}{2} \frac{\frac{1}{4\sqrt{(1.5+0.2)^3}}}{\frac{1}{2\sqrt{1.5+0.2}} + \frac{1}{10}} \approx 0.11.$$

I.e., the error in iteration $k+1$ is not the square of the error in the previous iteration, and this multiplied by 0.11.

Also discuss what happens when we start from $x_0 = 10$. In that case, the output is

$$\begin{aligned} x_0 &= 10 \\ x_1 &= -0.7053801508832727 \\ x_2 &= nan \\ x_3 &= nan \end{aligned}$$

This is due to the fact that in the second iteration, we need to evaluate $f(x_1)$ and $f'(x_1)$, but this isn't defined for x_1 due to the square root.