Hand in 2

To use the equations in Newtons method all of the equations were rearranged to be equal to 0. Thereafter, the equations and constants were written into a function like this.

VecDoub vecfunc(VecDoub\_I x)

{

    // n and d are held constant

    double d = 30;

    // std::cout << "n\_var" << n\_var << std::endl;

    //  x consists of {L\_0, L, p, x, theta, phi, a, H}

    double L\_0 = x[0];

    double L = x[1];

    double p = x[2];

    double x\_ = x[3];

    double theta = x[4];

    double phi = x[5];

    double a = x[6];

    double H = x[7];

    // material constants

    double v = 120;

    double k = 2.5;

    double w = 4.0;

    double alpha = 2 \* 10e-7;

    VecDoub result(8);

    result[0] = a \* (cosh(x\_ / a) - 1) - p;

    result[1] = 2 \* a \* sinh(x\_ / a) - L;

    result[2] = 2 \* x\_ + 2 \* k \* cos(theta) - d;

    result[3] = p + k \* sin(theta) - n\_var;

    result[4] = sinh(x\_ / a) - tan(phi);

    result[5] = (1 + v / (w \* L\_0)) \* tan(phi) - tan(theta);

    result[6] = L\_0 \* (1 + alpha \* H) - L;

    result[7] = (w \* L\_0) / (2 \* sin(phi)) - H;

    return result;

}

Then the main() function was set up like this, where I loop through the different n values, and find the solutions for each.

int main()

{

    // matrix for storing L\_0 and H for each n

    MatDoub L\_0\_H(6, 2);

    // initial guess;

    // x consists of {L\_0, L, p, x, theta, phi, a, H}

    VecDoub\_IO x\_guess(8);

    x\_guess[0] = 24;

    x\_guess[1] = 24;

    x\_guess[2] = 1;

    x\_guess[3] = 12;

    x\_guess[4] = 0.1;

    x\_guess[5] = 0.1;

    x\_guess[6] = 100;

    x\_guess[7] = 100;

    std::vector<double> n\_vec = {0.1, 0.2, 0.5, 1.0, 2.0, 5.0};

    bool check = true;

    /\* --------------------------------- n = 0.1 -------------------------------- \*/

    for (size\_t i = 0; i < n\_vec.size(); i++)

    {

        n\_var = n\_vec[i];

        cout << GREEN << "n = " << n\_vec[i] << RESET << endl;

        VecDoub\_IO x = x\_guess;

        newto(x, check, vecfunc);

        util::print(x, "solution\_x");

        L\_0\_H[i][0] = x[0];

        L\_0\_H[i][1] = x[7];

    }

    // loop through the matrix and print the values with corresponding n

    cout << YELLOW << setw(5) << "n" << setw(15) << "L\_0" << setw(15) << "H" << RESET << endl;

    for (size\_t i = 0; i < L\_0\_H.nrows(); i++)

    {

        cout << setw(5) << n\_vec[i] << setw(15) << L\_0\_H[i][0] << setw(15) << L\_0\_H[i][1] << endl;

    }

    return 0;

}

The defines for the colors can be seen here

// ANSI color codes

#define RESET "\033[0m"

#define RED "\033[31m"

#define GREEN "\033[32m"

#define YELLOW "\033[33m"

#define BLUE "\033[34m"

#define MAGENTA "\033[35m"

#define CYAN "\033[36m"

#define WHITE "\033[37m"

This gives the following values for L\_0 and H

A screenshot of a computer

Description automatically generated