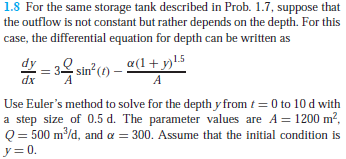
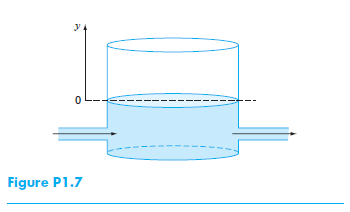
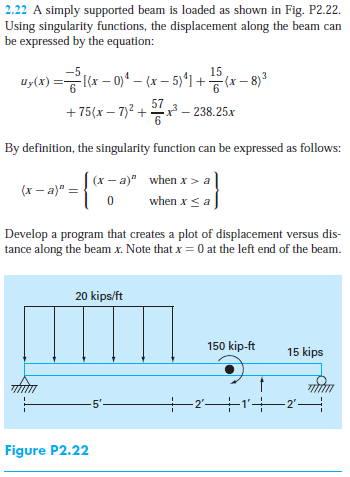
For this assignment I have all the output and code in the same console application. I forgot to check with the instructor on if this was ok, but the output and code for all programming problems are together.



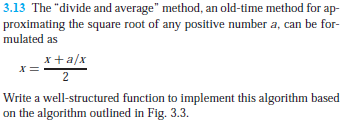
Solution: See code and output below

Special Instructions:

Write a program using your favorite high-level language (C++, VBA/Excel, VBA/CALC, etc.) to produce a table of values of uy(x) for 0 < x < 10 ft, with increment ∆x = 0.5 ft.  Using the results from the table produce the required graph by hand or using Excel or CALC.

Solution:

See table in output below (excel sheet also attached)

Special Instructions:

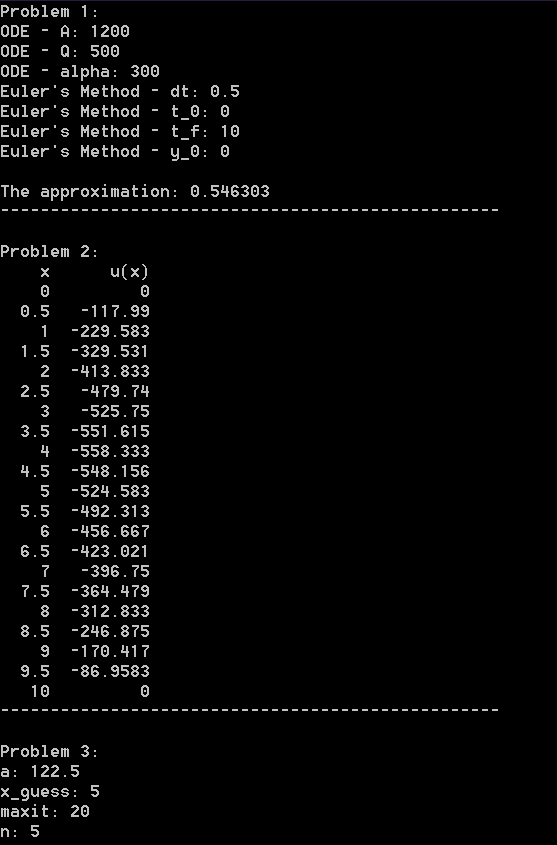
The program must take as input the following values:

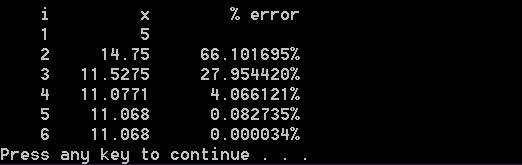
* The value whose square root is sought, a
* An initial guess of the solution, xguess
* The number of decimals, n, in order to calculate the error criteria, εs, according to equation (3.7)
* The maximum number of iterations allowed before declaring a diverging solution, maxit

The program should produce, as output, the value of the error criteria, εs, and a table showing the different iterations required to find a solution using the required error criteria. Show the results of your program for a = 122.5, xguess = 5.0, n = 5, maxit = 20.

Solution: See output below

Output:





main.cpp:

#include <iostream>

#include <iomanip>

#include <math.h>

#include "functions.h"

#include "..\shared\methods.cpp"

int main() {

// Problem 1

const double

A = 1200,

Q = 500,

alpha = 300,

Dt = 0.5,

t\_init = 0,

t\_final = 10,

y\_init = 0;

std::unordered\_map<double, double> result = Eulers(

[A, Q, alpha, Dt, t\_init, t\_final, y\_init](double t, double f) -> double {

double sin\_t = sin(t);

return 3 \* (Q / A) \* sin\_t \* sin\_t - (alpha \* pow(1 + f, 1.5) / A);

},

y\_init, t\_init, t\_final, Dt);

std::cout

<< "Problem 1: " << std::endl

<< "ODE - A: " << A << std::endl

<< "ODE - Q: " << Q << std::endl

<< "ODE - alpha: " << alpha << std::endl

<< "Euler's Method - dt: " << Dt << std::endl

<< "Euler's Method - t\_0: " << t\_init << std::endl

<< "Euler's Method - t\_f: " << t\_final << std::endl

<< "Euler's Method - y\_0: " << y\_init << std::endl << std:: endl

<< "The approximation: " << result[10] << std::endl

<< "--------------------------------------------------" << std::endl << std::endl;

// Problem 2

const double

x\_initial = 0,

x\_final = 10,

x\_increment = 0.5;

std::cout

<< "Problem 2:" << std::endl

<< std::setw(5) << "x" << std::setw(10) << "u(x)" << std::endl;

for (double x = 0; x <= 10; x += 0.5) {

std::cout

<< std::setw(5) << x

<< std::setw(10) << displacement(x)

<< std::endl;

}

std::cout

<< "--------------------------------------------------" << std::endl << std::endl;

// Problem 3

std::streamsize def\_precision = std::cout.precision();

int i = 1, n, max\_iterations;

double x, a, old, error = 100;

std::cout << "Problem 3:" << std::endl

<< "a: ";

std::cin >> a;

std::cout

<< "x\_guess: ";

std::cin >> x;

std::cout

<< "maxit: ";

std::cin >> max\_iterations;

std::cout

<< "n: ";

std::cin >> n;

double error\_criteria = 0.5 \* pow(10, 2 - n);

std::cout << std::endl

<< std::setw(5) << "i" << std::setw(10) << "x" << std::setw(15) << "% error" << std::endl

<< std::setw(5) << i << std::setw(10) << x << std::endl;

for (i = 2; i < max\_iterations && error > error\_criteria; ++i) {

old = x;

x = (x + (a / x)) / 2;

error = abs((x - old) / x) \* 100;

std::cout

<< std::setw(5) << i

<< std::setw(10) << x

<< std::setw(14) << std::fixed << std::setprecision(6) << error << "%"

<< std::endl << std::defaultfloat << std::setprecision(def\_precision);

}

system("pause");

return 0;

}

functions.h:

#pragma once

#include <math.h>

double displacement(double);

double singularity(double, double, double);

functions.cpp:

#include "functions.h"

double displacement(double x) {

return

(-5 \* (singularity(x, 0, 4) - singularity(x, 5, 4)) / 6) +

(15 \* singularity(x, 8, 3) / 6) +

(75 \* singularity(x, 7, 2)) +

(57 \* pow(x, 3) / 6) -

(238.25 \* x);

}

double singularity(double x, double a, double n) {

if (x <= a) { return 0; }

return pow(x - a, n);

}

methods.h:

#pragma once

#include <unordered\_map>

#include <functional>

std::unordered\_map<double, double> Eulers(std::function<double(double, double)>, double, double, double, double);

methods.cpp:

#include "methods.h"

std::unordered\_map<double, double> Eulers(

std::function<double(double, double)> ode,

double f\_init,

double t\_init,

double t\_final,

double t\_inc) {

double

t = t\_init,

f = f\_init;

std::unordered\_map<double, double> steps;

steps.insert(std::make\_pair(t, f));

do {

f += t\_inc \* ode(t, f);

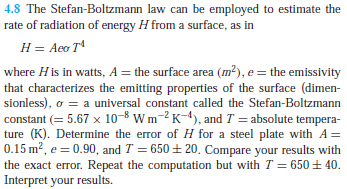
t += t\_inc;

steps.insert(std::make\_pair(t, f));

} while (t < t\_final);

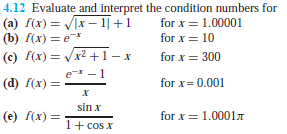
return steps;

}



Solution:

C:\Users\John\Desktop\render.cgi.png

Special instructions: just do part e

Solution:

I punched it into my calculator, and I get: -10,000.86