Hand in 4

BE AWARE THAT A LOT OF OPERATORS HAVE BEEN OVERLOADED IN utilities.h TO PROVIDE UTILITIES FOR USING BASIC OPERATORS WITH SCALARS AND VECTORS

Exercise i)

The differential equations are implemented as follows

```
VecDoub diff_eqs(Doub x, VecDoub_I y)
{
    VecDoub dydx(3);
    dydx[0] = exp(-x) * cos(y[1]) + pow(y[2], 2) - y[0];
    dydx[1] = cos(pow(y[2], 2)) - y[1];
    dydx[2] = cos(x) * exp(-pow(y[0], 2)) - y[2];
    return dydx;
}
int main(){
    util::title("exercise i");
    // start conditions
    double v1_0 = 1, v2_0 = 2, v3_0 = 3;
    VecDoub y(3);
    y[0] = v1_0;
    y[1] = v2_0;
    y[2] = v3_0;

    // get values of primes at start conditions
    double x = 0;
    VecDoub dxdy(3);
    dxdy = diff_eqs(x, y);
    cout << "dxdy: " << dxdy[0] << " " << dxdy[1] << " " << dxdy[2] <<
endl;
}
```

The output is:

```
exercise i
dxdy: 7.58385 -2.91113 -2.63212
```

Exercise ii)

The trapezoidal function is implemented with the code below:

```
VecDoub diff_eqs(Doub x, VecDoub_I y)
{
    VecDoub dydx(3);
    dydx[0] = exp(-x) * cos(y[1]) + pow(y[2], 2) - y[0];
    dydx[1] = cos(pow(y[2], 2)) - y[1];
    dydx[2] = cos(x) * exp(-pow(y[0], 2)) - y[2];
    return dydx;
}

void print_header_trapz()
{
    cout << setw(5) << "n" << setw(15) << "v1" << setw(15) << "v2" <<
setw(15) << "v3" << setw(15) << "error1" << setw(15) << "error2" <<
setw(15) << "error3" << endl;
}
// convert all nan to infinities in vec
void nan_to_inf(VecDoub_IO vec)
{
    for (int n = 0; n < vec.size(); n++)
    {
        if (isnan(vec[n]))
            vec[n] = INFINITY;
    }
}

void trapezoidal(VecDoub_IO &y, double x_start, const double x_des,
const double h)
{
    newt_struct my_newt_struct;
    my_newt_struct.y_init = y;
    my_newt_struct.x = x_start;
    my_newt_struct.h = h;
    VecDoub y_star(3);
    VecDoub dxdy(3);
    bool check = true;
    while (my_newt_struct.x < x_des)
    {
        auto bound_newt_eqs = bind(&newt_struct::trapz_eqs,
my_newt_struct, placeholders::_1);
        // euler step
        dxdy = diff_eqs(my_newt_struct.x, my_newt_struct.y_init);
        y_star = my_newt_struct.y_init + dxdy * h;
```

```
        // trapezoidal step
        newt(y_star, check, bound_newt_eqs);
        my_newt_struct.x += h;
        my_newt_struct.y_init = y_star;
        y = y_star;
    }
}
void trapezoidal_print(VecDoub_I y_start, const double x_start, const
double x_des, const double tol = 1e-10)
{
    VecDoub y(3, INFINITY);
    double h;
    double n = 50;
    int max_iter = 5;
    int i = 1;

    // for calculating alp_k and error
    VecDoub y_m1(3, INFINITY);
    VecDoub y_m2(3, INFINITY);
    VecDoub alp_k(3, INFINITY);
    VecDoub error(3, INFINITY);

    // print header
    print_header_trapz();
    while (i <= max_iter && (error[0] > tol || error[1] > tol ||
error[2] || tol))
    {
        y = y_start;
        h = (x_des - x_start) / n;
        trapezoidal(y, x_start, x_des, h);

        if (i > 3) // we need at least 3 iterations to calculate alp_k
        {
            alp_k = (y_m2 - y_m1) / (y_m1 - y);
            error = (y - y_m1) / (alp_k - 1.0);
        }
        cout << setw(5) << n << setw(15) << y[0] << setw(15)
             << setw(15) << y[1] << setw(15) << y[2] << setw(15) <<
error[0] << setw(15) << error[1] << setw(15) << error[2] << endl;
        y_m2 = y_m1;
        y_m1 = y;
        n *= 2;
        i++;

        // if error is nan set to infinity
        nan_to_inf(error);
```

```
    }
}
```

The following code is used in main()

```
    util::title("exercise ii");

    double x_start = 0;
    double x_des = 5;
    double tol = 1e-40;
    trapezoidal_print(y, x_start, x_des, tol);
```

The output can be seen here:

```
exercise ii
    n           v1          v2          v3
   50     0.220363    0.974731   -0.205817
  100     0.229308    0.973474   -0.233817
  200     0.237821    0.972203   -0.260788
  400     0.237843    0.972202     -0.2608
  800     0.236808     0.97236    -0.25749
```

Exercise iii)

The error was calculated by using Richardson extrapolation. This was achieved by implementing the general formulas with the following code:

```
alp_k = (y_m2 - y_m1) / (y_m1 - y);
error = (y - y_m1) / (alp_k - 1.0);
```

This gives an error of

error1	error2	error3
0.00101279	-0.000156869	-0.00329812