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:

A number on a black background

Description automatically generated

# 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:

A screenshot of a computer screen

Description automatically generated

# 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



