Common Code

I created a decimal class to avoid rounding errors with floating point keys for std::map.

decimal.hpp:

```
#pragma once
#include <iostream>
#include <algorithm>
#include <type_traits>
#include <stdint.h>
#include <vector>

class decimal;
namespace dec { int pow10(int); }
std::istream& operator>>(std::istream&, decimal&);
std::ostream& operator<<(std::ostream&, const decimal&);

class decimal {
private:
    typedef int64_t v_type;
    typedef unsigned int r_type;

    v_type value;
    r_type radix;

public:
    decimal() : value(0), radix(0) {}
    decimal(const decimal& obj) { value = obj.value; radix = obj.radix; }
    decimal& operator=(const decimal& obj) { value = obj.value; radix =
obj.radix; return *this; }

    r_type getRadix() const { return radix; }
    void setRadix(r_type);

    // Primitive Type Conversions
    template <typename N, typename
std::enable_if<std::is_arithmetic<N>::value>::type* = nullptr>
    explicit operator N() const { return (N)value / dec::pow10(radix); }

    template <typename N, typename
std::enable_if<std::is_arithmetic<N>::value>::type* = nullptr>
    decimal(const N& n, int r) { value = (n * dec::pow10(r + 1) + 5) / 10;
radix = r; }

    template <typename N, typename
std::enable_if<std::is_arithmetic<N>::value>::type* = nullptr>
    decimal(const N& n) {
        value = (int)n;
        radix = 0;

        // Determine an appropriate radix
        N r = (n - value) * 10;
        int temp = (int)r;
        while(temp != 0) {
```

```
            ++radix;
            value = (value * 10) + temp;
            r = (r - temp) * 10;
            temp = (int)r;
        }
    }

    // Comparison operators
    bool operator==(const decimal& n) const { decimal t = *this - n; return
t.value == 0; }
    bool operator!=(const decimal& n) const { decimal t = *this - n; return
t.value != 0; }
    bool operator< (const decimal& n) const { decimal t = *this - n; return
t.value <  0; }
    bool operator<=(const decimal& n) const { decimal t = *this - n; return
t.value <= 0; }
    bool operator> (const decimal& n) const { decimal t = *this - n; return
t.value >  0; }
    bool operator>=(const decimal& n) const { decimal t = *this - n; return
t.value >= 0; }

    // Arithmetic Operators
    friend decimal operator+(const decimal& l, const decimal& r) { return
decimal(l) += r; }
    friend decimal operator-(const decimal& l, const decimal& r) { return
decimal(l) -= r; }

    // Compound Arithmetic Operators
    decimal& operator+=(const decimal&);
    decimal& operator-=(const decimal&);

    // Multiplicative Operators
    friend decimal operator*(const decimal& l, const decimal& r) { return
(double)l * (double)r; }
    friend decimal operator/(const decimal& l, const decimal& r) { return
(double)r * (double)r; }

    // Compound Multiplicative Operators
    decimal& operator*=(const decimal& n) { *this = *this * n; return *this;
}
    decimal& operator/=(const decimal& n) { *this = *this / n; return *this;
}
};
```

decimal.cpp:

```
#include "decimal.hpp"

int dec::pow10(int pow) {
    static std::vector<int> list{1, 10, 100, 1000, 10000, 100000, 1000000};
    static int n = list.size() - 1;
    while (n < pow) {
        list.push_back(10 * list[n - 1]); ++n;
    }
    return list[pow];
}
```

```
std::istream& operator>>(std::istream& in, decimal& obj) {
    double d;
    in >> d;
    obj = d;
    return in;
}

std::ostream& operator<<(std::ostream& out, const decimal& obj) {
    out << (double)obj;
    return out;
}

void decimal::setRadix(r_type r) {
    if (r > radix) { value *= dec::pow10(r - radix); radix = r; }
    if (r < radix) { value = ((value / dec::pow10(radix + 1 - r)) + 5) / 10;
radix = r; }
}


decimal& decimal::operator+= (const decimal& n) {
    if (n.radix > radix) { setRadix(n.radix); }
    value += n.value * dec::pow10(radix - n.radix);
    return *this;
}
decimal& decimal::operator-=(const decimal& n) {
    if (n.radix > radix) { setRadix(n.radix); }
    value -= n.value * dec::pow10(radix - n.radix);
    return *this;
}
```

Problem 1

25.21 The logistic model is used to simulate population as in

$$\frac{dp}{dt} = k_{gm}\,(1 - p/p_{\max})\,p$$

where p = population, k_{gm} = the maximum growth rate under unlimited conditions, and $p_{\max}$ = the carrying capacity. Simulate the world's population from 1950 to 2000 using one of the numerical methods described in this chapter. Employ the following initial conditions and parameter values for your simulation: p_0 (in 1950) = 2555 million people, k_{gm} = 0.026/yr, and $p_{\max}$ = 12,000 million people. Have the function generate output corresponding to the dates for the following measured population data. Develop a plot of your simulation along with the data.

t	1950	1960	1970	1980	1990	2000
p	2555	3040	3708	4454	5276	6079

```
Problem 1:
t          p          p_exact    p_actual
-----------------------------------------
1950       2555       2555       2555
1960       3116.62    3040       3040
1970       3752.64    3708       3708
1980       4453.39    4454       4454
1990       5202.45    5276       5276
2000       5977.7     6079       6079
```

Your table of results show the values of population, p, predicted for the years, 1950 to 2000, by solving the ODE shown above using the Runge-Kutta 4th-order method. The table of results should also show the values predicted by the exact solution to the ODE (see below), as well as the actual population values shown in the table describing problem 25.21 as shown above.

Code: (notice my RK4 method is the same in each c++ problem, I just change the three typedefs)

```cpp
#include "a6p1.hpp"
#include <iomanip>
#include <iostream>
#include <functional>
#include <map>
#include "../shared/decimal.hpp"

typedef std::function<double(const decimal&, const double&)> ODE;
typedef std::map<decimal, double> Table;
typedef std::pair<decimal, double> Point;

Table RK4(ODE f, Point init, decimal t_final, double h) {
    double k1, k2, k3, k4,
        s = init.second;
    decimal t = init.first;

    Table data{init};
    do {
        k1 = f(t, s);
        k2 = f(t + h / 2, s + k1 * h / 2);
        k3 = f(t + h / 2, s + k2 * h / 2);
        k4 = f(t + h, s + k3 * h);

        s += h * (k1 + 2 * (k2 + k3) + k4) / 6;
        t += h;

        data.insert(std::make_pair(t, s));
    } while (t < t_final);

    return data;
}
```

```cpp
void assign6::Prob1() {
    Table exact{
        {1950, 2555},
        {1960, 3040},
        {1970, 3708},
        {1980, 4454},
        {1990, 5276},
        {2000, 6079},
    };
    Table actual{
        {1950, 2555},
        {1960, 3040},
        {1970, 3708},
        {1980, 4454},
        {1990, 5276},
        {2000, 6079},
    };

    const double kgm = 0.026;
    const double pmax = 12000; // p measured in millions
    ODE dpdt = [&kgm, &pmax](const decimal& t, const double& p) {
        return kgm * (1 - p / pmax) * p;
    };

    Table results = RK4(dpdt, std::make_pair(1950, 2555), 2000, 0.1);

    std::cout << "Problem 1: " << std::left << std::endl
        << std::setw(10) << "t"
        << std::setw(10) << "p"
        << std::setw(10) << "p_exact"
        << std::setw(10) << "p_actual"
        << std::endl
        << "----------------------------------------" << std::endl;
    for (decimal year = 1950; year <= 2000; year += 10) {
        std::cout
            << std::setw(10) << (double)year
            << std::setw(10) << results[year]
            << std::setw(10) << exact[year]
            << std::setw(10) << actual[year]
            << std::endl;
    }
    std::cout << std::endl;
}
```

Problem 2

Assignment 6 - Problem [2]: Prepare a SCILAB script, similar to the one shown above, to solve problem 25.21, i.e., the same problem solved as Problem [1] for this Assignment:

25.21 The logistic model is used to simulate population as in

$$\frac{dp}{dt} = k_{gm}\,(1 - p/p_{max})\,p$$

where p = population, k_{gm} = the maximum growth rate under unlimited conditions, and p_{max} = the carrying capacity. Simulate the world's population from 1950 to 2000 using one of the numerical methods described in this chapter. Employ the following initial conditions and parameter values for your simulation: p_0 (in 1950) = 2555 million people, k_{gm} = 0.026/yr, and p_{max} = 12,000 million people. Have the function generate output corresponding to the dates for the following measured population data. Develop a plot of your simulation along with the data.

t	1950	1960	1970	1980	1990	2000
p	2555	3040	3708	4454	5276	6079

Produce a plot showing the numerical solution for the values of t shown in the table given in Problem 25.21, together with the exact solution:

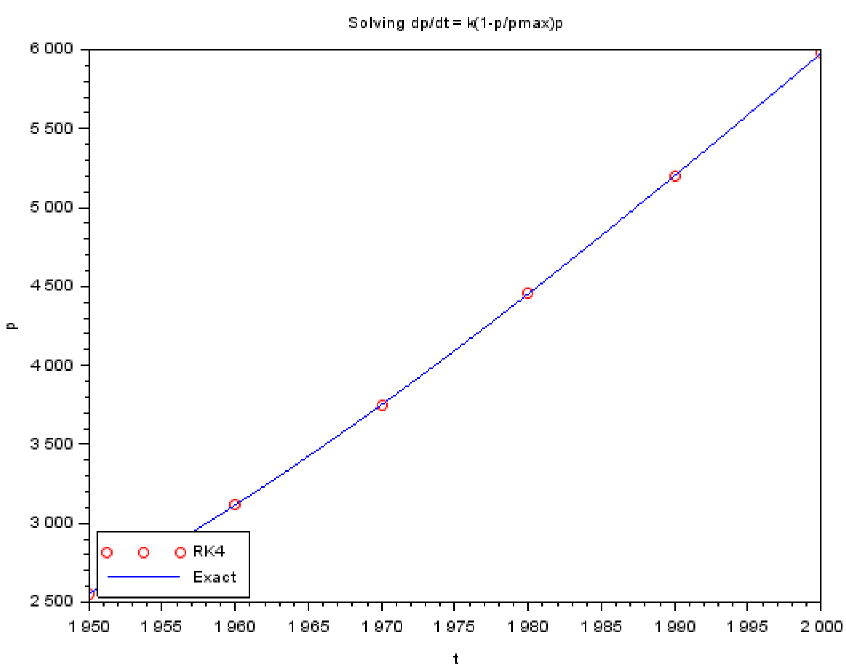
$$p(t) := \frac{p_{max}}{1 - \left(1 - \frac{p_{max}}{p_0}\right) \cdot e^{-k_{gm} \cdot \left(t - t_0\right)}}$$

```
-->exec('C:\Users\John\Source\USU\ENGR 2450\src\assign6\scilab\Prob2.sci', -1)

 x   y

 --------

    1950.     2555.
    1960.     3116.6194
    1970.     3752.6407
    1980.     4453.3888
    1990.     5202.448
    2000.     5977.7003
```



Code:

```
function [z] = f(x, y)
  z = .026 * (1 - y / 12000.0) * y;
endfunction;

x0 = 1950.0; y0 = 2555.0; xn = 2000.0; h = 10.0;
x = [x0:h:xn];
y = ode("rk", y0, x0, x, f);

disp("x   y"); // show titles for output table
disp("--------"); // show a line for the table
disp([x' y']);

// fe(x) = exact solution
function z = fe(x)
  z = 12000 / (1 - (1 - (12000 / 2555.0)) * exp(-.026 * (x - 1950.0)));
endfunction;

xe = [x0:h / 10:xn];
n = length(xe);
for i = 1:n
  ye(i) = fe(xe(i));
end;

// NEXT: plot numerical solution as red circles and
// exact solution as a continuous blue line
plot(x, y, 'ro', xe, ye, '-b');
legend('RK4', 'Exact', 3);
xtitle('Solving dp/dt = k(1-p/pmax)p', 't', 'p');
```

Problem 3

Assignment 6 - Problem [3]: Prepare a SCILAB script, similar to the one shown above, to solve problem 25.17, shown below:

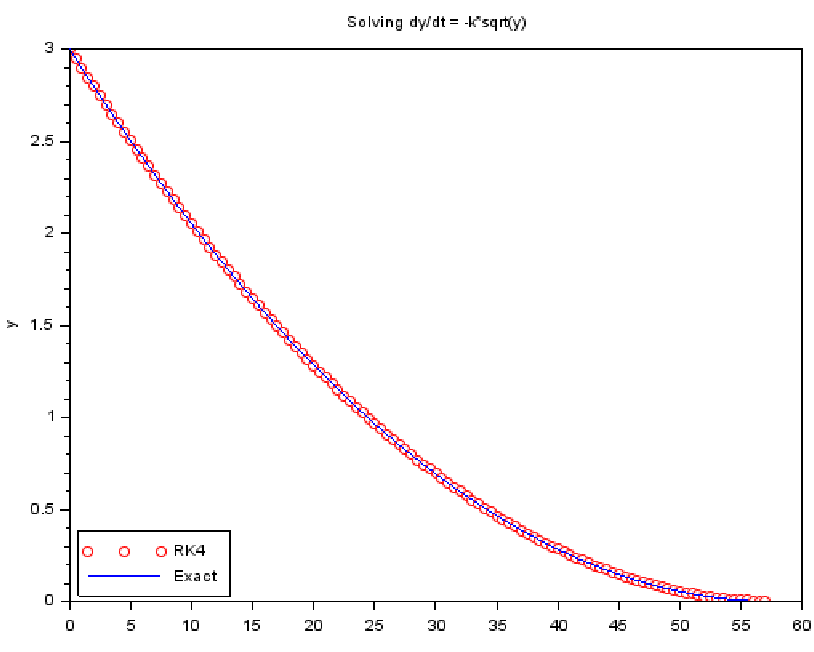
> **25.17** If water is drained from a vertical cylindrical tank by opening a valve at the base, the water will flow fast when the tank is full and slow down as it continues to drain. As it turns out, the rate at which the water level drops is:
>
> $$\frac{dy}{dt} = -k\sqrt{y}$$
>
> where k is a constant depending on the shape of the hole and the cross-sectional area of the tank and drain hole. The depth of the water y is measured in meters and the time t in minutes. If $k = 0.06$, determine how long it takes the tank to drain if the fluid level is initially 3 m. Solve by applying Euler's equation and writing a computer program or using Excel. Use a step of 0.5 minutes.

Notes: (1) Do not solve this problem using the Euler's method as indicated in the problem statement. Instead, use a script similar to the one used in this document. Select a vector of values of t = [0, 0.5, 1.0, ..., tmax]. Guess the value of tmax so that your numerical solution shows that y becomes zero for some time t in the range 0 < t < tmax.

(2) Use the following exact solution:

$$y = \frac{k^2}{4}\cdot\left(\frac{2}{k}\cdot\sqrt{y_0} - t\right)^2$$

(3) In your assignment report show the SCILAB script you used to solve the problem, the table of numerical results, and the graph showing the numerical and the



```
--> exec('C:\Users\John\Source\USU\ENGR
2450\src\assign6\scilab\Prob3.sci', -1)

 x  y
 --------
    0.      3.
    0.5     2.9482635
    1.      2.896977
    1.5     2.8461404
    2.      2.7957539
    2.5     2.7458174
    3.      2.6963309
    3.5     2.6472943
    4.      2.5987078
    4.5     2.5505713
    5.      2.5028848
    5.5     2.4556482
    6.      2.4088617
    6.5     2.3625252
    7.      2.3166387
    7.5     2.2712021
    8.      2.2262156
    8.5     2.1816791
    9.      2.1375926
    9.5     2.093956
    10.     2.0507695
    10.5    2.008033
    11.     1.9657465
    11.5    1.9239099
    12.     1.8825234
    12.5    1.8415869
    13.     1.8011004
    13.5    1.7610638
    14.     1.7214773
    14.5    1.6823408
    15.     1.6436543
    15.5    1.6054177
    16.     1.5676312
    16.5    1.5302947
    17.     1.4934082
    17.5    1.4569717
    18.     1.4209851
    18.5    1.3854486
    19.     1.3503621
    19.5    1.3157256
    20.     1.281539
    20.5    1.2478025
    21.     1.214516
    21.5    1.1816795
    22.     1.1492929
    22.5    1.1173564
    23.     1.0858699
    23.5    1.0548334
    24.     1.0242468
    24.5    0.9941103
    25.     0.9644238
    25.5    0.9351873
    26.     0.9064007
    26.5    0.8780642
    27.     0.8501777
    27.5    0.8227412
    28.     0.7957546
    28.5    0.7692181
    29.     0.7431316
    29.5    0.7174951
    30.     0.6923085
    30.5    0.6675720
    31.     0.6432855
    31.5    0.6194490
    32.     0.5960624
    32.5    0.5731259
    33.     0.5506394
    33.5    0.5286029
    34.     0.5070164
    34.5    0.4858798
    35.     0.4651933
    35.5    0.4449568
    36.     0.4251703
    36.5    0.4058337
    37.     0.3869472
    37.5    0.3685107
    38.     0.3505242
    38.5    0.3329876
    39.     0.3159011
    39.5    0.2992646
    40.     0.2830781
    40.5    0.2673415
    41.     0.2520550
    41.5    0.2372185
    42.     0.2228320
    42.5    0.2088954
    43.     0.1954089
    43.5    0.1823724
    44.     0.1697859
    44.5    0.1576493
    45.     0.1459628
    45.5    0.1347263
    46.     0.1239398
    46.5    0.1136032
    47.     0.1037167
    47.5    0.0942802
    48.     0.0852937
    48.5    0.0767571
    49.     0.0686706
    49.5    0.0610341
    50.     0.0538476
    50.5    0.0471111
    51.     0.0408245
    51.5    0.0349880
    52.     0.0296015
    52.5    0.0246650
    53.     0.0201784
    53.5    0.0161419
    54.     0.0125554
    54.5    0.0094189
    55.     0.0067323
    55.5    0.0044958
    56.     0.0027093
    56.5    0.0013728
    57.     0.0004862
```

Code:

```
function [z]=f(x, y)
  z = -0.06 * sqrt(y);
endfunction;

x0 = 0.0; y0 = 3.0; xn = 57; h = 0.5;
x = [x0:h:xn];
y = ode("rk", y0, x0, x, f);

disp("x   y"); // show titles for output table
disp("--------"); // show a line for the table
disp([x' y']);

// fe(x) = exact solution
function z=fe(x)
  z = (0.06)^2 / 4 * (2 / 0.06 * sqrt(3.0) - x)^2;
endfunction;

xe = [x0:h / 10:xn];
n = length(xe);
for i = 1:n
  ye(i) = fe(xe(i));
end;

// NEXT: plot numerical solution as red circles and
// exact solution as a continuous blue line
plot(x, y, 'ro', xe, ye, '-b');
legend('RK4', 'Exact', 3);
xtitle('Solving dy/dt = -k*sqrt(y)', 't', 'y');
```

Problem 4

Assignment 6 - Problem [4]

Using your favorite computer language (VBA/Excel, C++) implement the solution to the Lorenz equations, shown earlier, using the pseudocode of Figure 25.18 (see above). The Lorenz equations are shown here, once more:

> An example of a simple model based on atmospheric fluid dynamics is the *Lorenz equations* developed by the American meteorologist Edward Lorenz,
>
> $$\frac{dx}{dt} = -\sigma x + \sigma y \qquad (28.6)$$
>
> $$\frac{dy}{dt} = rx - y - xz \qquad (28.7)$$
>
> $$\frac{dz}{dt} = -bz + xy \qquad (28.8)$$
>
> Lorenz developed these equations to relate the intensity of atmospheric fluid motion, x, to temperature variations y and z in the horizontal and vertical directions, respectively.

For your solution, use the values given in the textbook, namely, σ = 10, b = 2.666667, and r = 28, and use the initial conditions x = y = z = 5. Solve the system of equations in the interval 0 < t < 20, with an increment dx = 0.1, and a printing interval of xout = 2.0.

In your assignment report show:
(a) Code and interface with input/output, if using VBA/Excel, or
    Code and printout of output screen if using C++
(b) Table of results showing x, y, z as functions of t

```
Problem 4:
t       x(t)            y(t)            z(t)
---------------------------------------------
0       5               5               5
2       -8.21676        -11.9054        20.6038
4       -7.75106        -11.6741        19.2247
6       -5.09585        -7.89709        16.4316
8       -1.59655        -1.02981        19.9723
10      3.97735         6.22195         20.4694
12      4.13094         5.16084         19.5005
14      3.03881         2.97829         20.5851
16      5.39265         0.00529396      30.1135
18      4.45528         7.54385         14.7979
20      11.3718         0.829441        39.3069
```

Code:

```
#include "a6p4.hpp"
#include <iomanip>
#include <iostream>
#include <functional>
#include <type_traits>
#include <map>

#include "../shared/decimal.hpp"
```

```
struct s3eqs {
    double x, y, z;

    s3eqs() : x(0), y(0), z(0) {}
    s3eqs(double sx, double sy, double sz) : x(sx), y(sy), z(sz) {}
    s3eqs(const s3eqs& s) { x = s.x; y = s.y; z = s.z; }
    template <typename N, typename
std::enable_if<std::is_arithmetic<N>::value>::type* = nullptr>
    s3eqs(const N& n) { x = n; y = n; z = n; }

    s3eqs& operator+= (const s3eqs& s) { x += s.x; y += s.y; z += s.z; return
*this; }
    s3eqs& operator-= (const s3eqs& s) { x -= s.x; y -= s.y; z -= s.z; return
*this; }
    s3eqs& operator*= (const s3eqs& s) { x *= s.x; y *= s.y; z *= s.z; return
*this; }
    s3eqs& operator/= (const s3eqs& s) { x /= s.x; y /= s.y; z /= s.z; return
*this; }
    friend s3eqs operator+ (const s3eqs& l, const s3eqs& r) { return s3eqs(l)
+= r; }
    friend s3eqs operator- (const s3eqs& l, const s3eqs& r) { return s3eqs(l)
-= r; }
    friend s3eqs operator* (const s3eqs& l, const s3eqs& r) { return s3eqs(l)
*= r; }
    friend s3eqs operator/ (const s3eqs& l, const s3eqs& r) { return s3eqs(l)
/= r; }
};

typedef std::function<s3eqs(const decimal&, const s3eqs&)> ODE;
typedef std::map<decimal, s3eqs> Table;
typedef std::pair<decimal, s3eqs> Point;

Table RK4(ODE f, Point init, decimal t_final, double h) {
    s3eqs k1, k2, k3, k4,
        s = init.second;
    decimal t = init.first;

    Table data{init};
    do {
        k1 = f(t, s);
        k2 = f(t + h / 2, s + k1 * h / 2);
        k3 = f(t + h / 2, s + k2 * h / 2);
        k4 = f(t + h, s + k3 * h);

        s += h * (k1 + 2 * (k2 + k3) + k4) / 6;
        t += h;

        data.insert(std::make_pair(t, s));
    } while (t < t_final);

    return data;
}
```

```cpp
void assign6::Prob4() {
    const double sigma = 10, b = 2.666667, r = 28;
    ODE dsdt = [&sigma, &b, &r](const decimal& t, const s3eqs& s) {
        s3eqs ds;
        ds.x = -sigma * s.x + sigma * s.y;
        ds.y = r * s.x - s.y - s.x * s.z;
        ds.z = -b * s.z + s.x * s.y;
        return ds;
    };

    Table results = RK4(dsdt, std::make_pair(0, 5), 20, 0.1);

    std::cout << "Problem 4: " << std::left << std::endl
        << std::setw(7) << "t"
        << std::setw(15) << "x(t)"
        << std::setw(15) << "y(t)"
        << std::setw(15) << "z(t)"
        << std::endl
        << "----------------------------------------------" << std::endl;
    for (decimal t = 0; t <= 20; t += 2) {
        std::cout
            << std::setw(7) << (double)t
            << std::setw(15) << results[t].x
            << std::setw(15) << results[t].y
            << std::setw(15) << results[t].z
            << std::endl;
    }
    std::cout << std::endl;
}
```

Problem 5

```
Assignment 6 - Problem [5]
Solve problem 28.19 from the Chapra/Canale, 6th-edition, textbook.
```

28.19 The following equation can be used to model the deflection of a sailboat mast subject to a wind force:

$$\frac{d^2y}{dz^2} = \frac{f}{2EI}(L-z)^2$$

where f = wind force, E = modulus of elasticity, L = mast length, and I = moment of inertia. Calculate the deflection if $y = 0$ and $dy/dz = 0$ at $z = 0$. Use parameter values of $f = 60$, $L = 30$, $E = 1.25 \times 10^8$, and $I = 0.05$ for your computation.

```
The conversion of this second-order ODE into a system of two, first-order, ODEs was shown
above. What you need to do is modify the code you developed for Problem [4], to solve the
resulting system of two, first-order, ODEs for this problem. The solution uses a domain
0 < z < L.  For your solution, use an increment dz = 0.5, and xout = 5.0

In your assignment report show:
(a) Code and interface with input/output, if using VBA/Excel, or
    Code and printout of output screen if using C++
(b) Table of results showing y, and dy/dz, as functions of z
```

Output:

```
Problem 5:
z       y'(z)      y''(z)
-------------------------
0       0          0
5       0.04825    0.0182
10      0.172      0.0304
15      0.34425    0.0378
20      0.544      0.0416
25      0.75625    0.043
30      0.972      0.0432
```

Code:

```
#include "a6p5.hpp"
#include <iomanip>
#include <iostream>
#include <functional>
#include <type_traits>
#include <math.h>
#include <map>

#include "../shared/decimal.hpp"
```

```cpp
struct s2eqs {
    double y1, y2;

    s2eqs() : y1(0), y2(0) {}
    s2eqs(double sx, double sy, double sz) : y1(sx), y2(sy) {}
    s2eqs(const s2eqs& s) { y1 = s.y1; y2 = s.y2; }
    template <typename N, typename std::enable_if<std::is_arithmetic<N>::value>::type* =
nullptr>
    s2eqs(const N& n) { y1 = n; y2 = n; }

    s2eqs& operator+= (const s2eqs& s) { y1 += s.y1; y2 += s.y2; return *this; }
    s2eqs& operator-= (const s2eqs& s) { y1 -= s.y1; y2 -= s.y2; return *this; }
    s2eqs& operator*= (const s2eqs& s) { y1 *= s.y1; y2 *= s.y2; return *this; }
    s2eqs& operator/= (const s2eqs& s) { y1 /= s.y1; y2 /= s.y2; return *this; }
    friend s2eqs operator+ (const s2eqs& l, const s2eqs& r) { return s2eqs(l) += r; }
    friend s2eqs operator- (const s2eqs& l, const s2eqs& r) { return s2eqs(l) -= r; }
    friend s2eqs operator* (const s2eqs& l, const s2eqs& r) { return s2eqs(l) *= r; }
    friend s2eqs operator/ (const s2eqs& l, const s2eqs& r) { return s2eqs(l) /= r; }
};

typedef std::function<s2eqs(const decimal&, const s2eqs&)> ODE;
typedef std::map<decimal, s2eqs> Table;
typedef std::pair<decimal, s2eqs> Point;

Table RK4(ODE f, Point init, decimal t_final, double h) {
    s2eqs k1, k2, k3, k4,
        s = init.second;
    decimal t = init.first;

    Table data{init};
    do {
        k1 = f(t, s);
        k2 = f(t + h / 2, s + k1 * h / 2);
        k3 = f(t + h / 2, s + k2 * h / 2);
        k4 = f(t + h, s + k3 * h);

        s += h * (k1 + 2 * (k2 + k3) + k4) / 6;
        t += h;

        data.insert(std::make_pair(t, s));
    } while (t < t_final);

    return data;
}


void assign6::Prob5() {
    const double f = 60, L = 30, E = 125000000, I = 0.05;
    ODE dsdt = [&f, &L, &E, &I](const decimal& z, const s2eqs& s) {
        s2eqs ds;
        double temp = L - (double)z;
        ds.y1 = s.y2;
        ds.y2 = f * temp * temp / (2 * E * I);
        return ds;
    };

    Table results = RK4(dsdt, std::make_pair(0, 0), L, 0.5);
```

```
    std::cout << "Problem 5: " << std::left << std::endl
        << std::setw(7) << "z"
        << std::setw(10) << "y'(z)"
        << std::setw(10) << "y''(z)"
        << std::endl
        << "-------------------------" << std::endl;
    for (decimal z = 0; z <= L; z += 5) {
        std::cout
            << std::setw(7) << (double)z
            << std::setw(10) << results[z].y1
            << std::setw(10) << results[z].y2
            << std::endl;
    }
    std::cout << std::endl;
}
```

Problem 6

Assignment 6 - Problem [6]

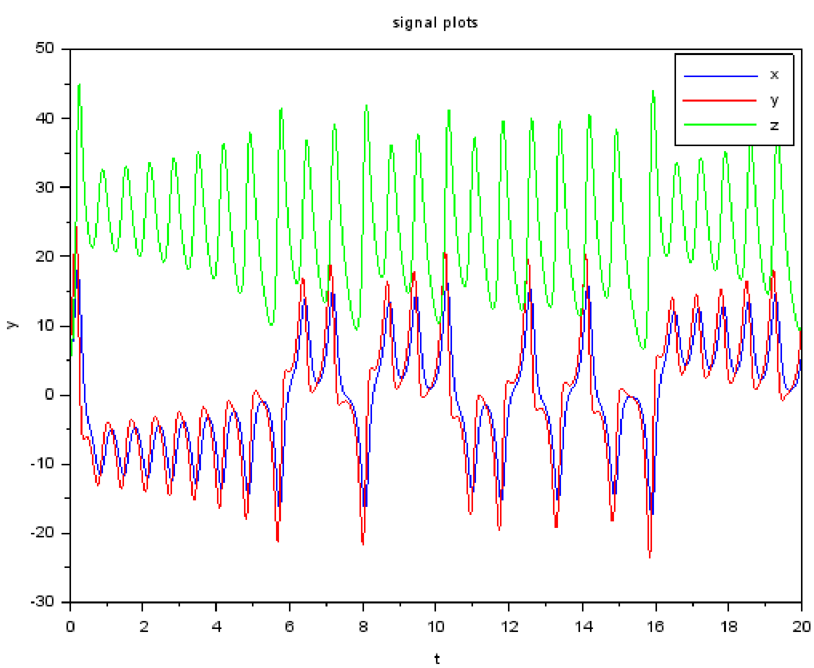
Modify the script shown above so as to produce the numerical solution of the Lorenz equations shown below:

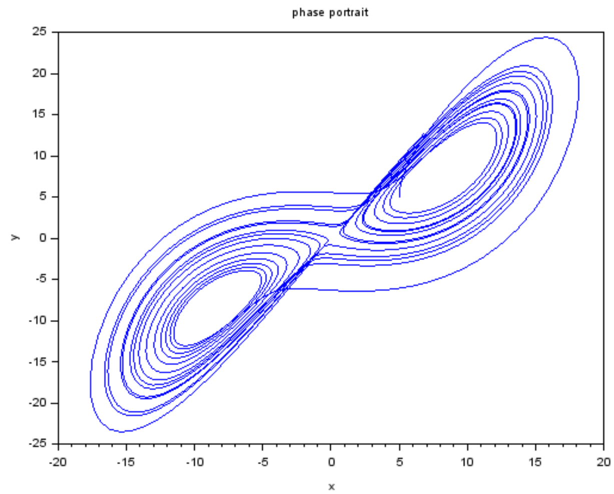
> An example of a simple model based on atmospheric fluid dynamics is the *Lorenz equations* developed by the American meteorologist Edward Lorenz,
>
> $$\frac{dx}{dt} = -\sigma x + \sigma y \qquad (28.6)$$
>
> $$\frac{dy}{dt} = rx - y - xz \qquad (28.7)$$
>
> $$\frac{dz}{dt} = -bz + xy \qquad (28.8)$$
>
> Lorenz developed these equations to relate the intensity of atmospheric fluid motion, x, to temperature variations y and z in the horizontal and vertical directions, respectively.

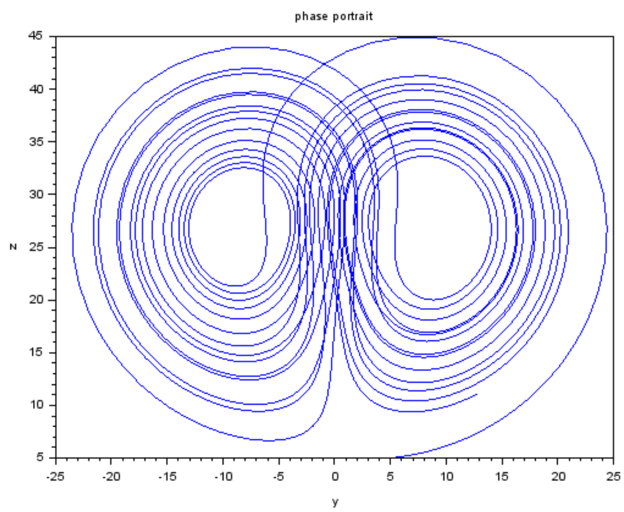
For your solution, use the values given in the textbook, namely, σ = 10, b = 2.666667, and r = 28, and use the initial conditions x = y = z = 5. Solve the system of equations in the interval 0 < t < 20, with an increment dt = 0.5, and produce a table of values of t, x, y, and z. Then, run the solution again, but now using an increment of dt = 0.01, and produce signal plots of x vs. t, y vs. t, and z vs. t, in the same set of axes. Also, produce phase portraits of x vs. y, y vs. z, and z vs. x, in separate plots. It is not necessary to report the solution for this second case (with dt = 0.01), only report the plots.

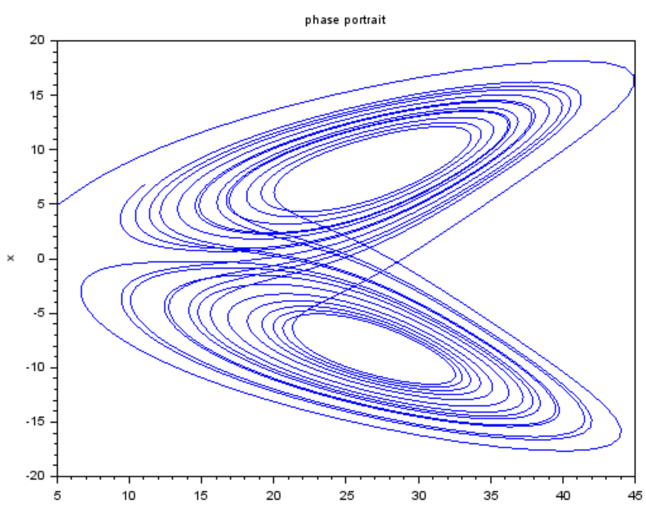
In your assignment report show the following:

(a) A copy of your script
(b) The results shown in the SCILAB script for dx = 0.5
(c) Printscreens of the single signal plot, and of the three phase portraits for dx = 0.01




phase portrait
y
x
25
20
15
10
5
0
-5
-10
-15
-20
-25
-20
-15
-10
-5
0
5
10
15
20


phase portrait
z
y
45
40
35
30
25
20
15
10
5
-25
-20
-15
-10
-5
0
5
10
15
20
25


phase portrait
x
z
20
15
10
5
0
-5
-10
-15
-20
5
10
15
20
25
30
35
40
45

```
-->exec('C:\Users\John\Source\USU\ENGR 2450\src\assign6\scilab\Prob6.sci', -1)

 t     x     y     z

    0.      5.           5.           5.
    0.5   - 3.7814106  - 6.3169311    23.647634
    1.    - 7.0906465  - 4.1386816    29.061623
    1.5   - 11.595625  - 10.143753    32.548029
    2.    - 9.1334883  - 12.695516    22.463797
    2.5   - 4.769982   - 6.0499785    20.033587
    3.    - 5.0081004  - 2.5352284    26.615529
    3.5   - 12.125545  - 8.7253588    35.088112
    4.    - 8.1282379  - 12.686092    18.56868
    4.5   - 2.3818654  - 2.7159269    18.774727
    5.    - 7.6106433  - 0.5349684    33.467966
    5.5   - 3.686293   - 6.7968469    10.088964
    6.      1.0242741    3.3178066    22.77626
    6.5     10.854489    4.497227     36.107139
    7.      6.1728431    10.650915    13.895474
    7.5     0.0907112  - 0.8721182    19.676757
    8.    - 14.232407  - 21.597105    25.62721
    8.5     4.9661488    7.8260625    16.949726
    9.      2.7805677    1.4514892    23.063958
    9.5     13.263849    8.3341082    37.837017
    10.     2.1147599    3.7251072    11.395483
    10.5    1.5377615  - 3.2983107    27.516312
    11.   - 14.092675  - 13.880206    34.713982
    11.5  - 2.801965   - 4.704838     13.16033
    12.   - 1.7012279    1.9638365    26.076456
    12.5    12.909464    19.531438    24.169917
    13.   - 1.9550934  - 3.2937689    15.004878
    13.5  - 4.4209484    1.7909131    30.334953
    14.     7.7547255    13.784561    13.699074
    14.5  - 1.6475387  - 2.9407104    18.197664
    15.   - 7.6171841  - 0.1606193    33.787152
    15.5  - 0.7267126  - 1.3565902    8.8388018
    16.   - 7.3987413    4.4236376    36.789302
    16.5    12.085237    12.815898    30.873117
    17.     7.2777553    10.622188    19.635627
    17.5    3.8369271    4.0400133    21.13478
    18.     6.8700656    2.157853     30.694365
    18.5    12.98711     16.074606    29.346399
    19.     3.5027921    5.5218223    15.163712
    19.5    2.4027939  - 0.5091207    25.361272
    20.     6.9793893    12.966543    11.232319
```

Code:

```
//Script to solve dY/dx = F(x, Y), Y(x0) = Y0
function [Z] = F(x, Y)
  Z(1)= -10 * Y(1) + 10 * Y(2);
  Z(2)= 28 * Y(1) - Y(2) - Y(1) * Y(3);
  Z(3)= -2.666667 * Y(3) + Y(1) * Y(2);
Endfunction;

//ODE solution
x = [0.0:0.01:20.0];
x0 = 0;
Y0 = [5;5;5];
Y = ode("rk", Y0, x0, x, F); //solve ODE

disp("t     x     y     z")
disp([x' Y']);
y1 = Y(1,:); y2 = Y(2,:); y3 = Y(3,:); //extract y1, y2, y3

scf();
plot(x, y1, 'b-', x, y2, 'r-', x, y3, 'g-');
legend('x', 'y', 'z', 1);
xtitle('signal plots', 't', 'y');

scf();
plot(y1, y2);
xtitle('phase portrait', 'x', 'y');

scf();
plot(y2, y3);
xtitle('phase portrait', 'y', 'z');

scf();
plot(y3, y1);
xtitle('phase portrait', 'z', 'x');
```

Problem 7

```
Assignment 6 - Problem [7]
Solve problem 28.47 (see below) by modifying the script shown above that uses function 'ode'
in SCILAB. First, you'll need to convert the second-order ODE described in the problem
statement, into a system of two, first-order, ODEs, in order to use the modified script.
```

28.47 A forced damped spring-mass system (Fig. P28.47) has the following ordinary differential equation of motion:

$$m\frac{d^2x}{dt^2} + a\left|\frac{dx}{dt}\right|\frac{dx}{dt} + kx = F_o \sin(\omega t)$$

where x = displacement from the equilibrium position, t = time, $m = 2$ kg mass, $a = 5$ N/(m/s)^2, and $k = 6$ N/m. The damping term is nonlinear and represents air damping. The forcing function $F_o \sin(\omega t)$ has values of $F_o = 2.5$ N and $\omega = 0.5$ rad/sec. The initial conditions are

Initial velocity $\quad \frac{dx}{dt} = 0$ m/s

Initial displacement $\quad x = 1$ m

Solve this equation using a numerical method over the time period $0 \le t \le 15$ s. Plot the displacement and velocity versus time, and plot the forcing function on the same curve. Also, develop a separate plot of velocity versus displacement.

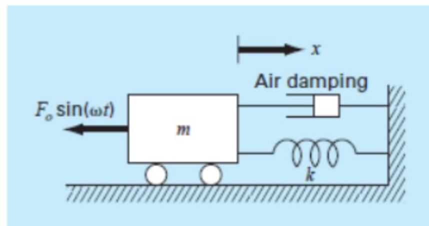


Figure P28.47

```
In your assignment report include the following:
(a) A write up showing how you converted the 2nd-order ODE into a system of two, 1st-order,
    ODEs.
(b) The SCILAB script used to produce the solution.
(c) The table of position x, and velocity dx/dt, as function of time t, produced by 'ode'
    (these results can be copied from the SCILAB console after the script is executed).
(d) The plot of x vs. t, dx/dt vs. t, required from the problem, in the same set of axes
    (i.e., signal plots).
(e) The phase portrait of dx/dt vs. x, required from the problem.
```

Write-up in Latex:

```
\\m{x}'' + a\left | {x}' \right |{x}' + kx = F_{_{0}}sin(\omega t)
\\2{x}'' + 5\left | {x}' \right |{x}' + 6x = 2.5sin(.5 t)
\\
\\Y_{_{1}} = x \ \ \ Y_{_{1}}(0) = 1
\\Y_{_{2}} = Y_{_{1}}' \ \ Y_{_{2}}(0) = 0
\\
\\Y_{_{2}}' = \frac{2.5sin(.5t) - 5\left | Y_{_{2}}' \right |Y_{_{2}}' - 6
Y_{_{1}}}{2}
```

$$mx'' + a\left|x'\right|x' + kx = F_0 sin(\omega t)$$
$$2x'' + 5\left|x'\right|x' + 6x = 2.5sin(.5t)$$

$$Y_1 = x \quad Y_1(0) = 1$$
$$Y_2 = Y_1' \quad Y_2(0) = 0$$

$$Y_2' = \frac{2.5sin(.5t) - 5\left|Y_2'\right|Y_2' - 6Y_1}{2}$$

```
-->exec('C:\Users\John\Source\USU\ENGR 2450\src\assign6\scilab\Prob7.sci', -1)
 t   Displacement   Velocity
    0.        1.           0.
    0.25      0.9154604  - 0.6166714
    0.5       0.7277995  - 0.8234252
    0.75      0.5241681  - 0.7814252
    1.        0.3443730  - 0.6485298
    1.25      0.2026036  - 0.4820589
    1.5       0.1047191  - 0.2986876
    1.75      0.0544165  - 0.1012159
    2.        0.0555627    0.1123009
    2.25      0.1081797    0.2971271
    2.5       0.1966896    0.3951189
    2.75      0.2985625    0.4077322
    3.        0.3956909    0.3617355
    3.25      0.4764842    0.2800503
    3.5       0.5338876    0.1761942
    3.75      0.5632914    0.0567133
    4.        0.5611603  - 0.0751275
    4.25      0.5265626  - 0.1968859
    4.5       0.4661165  - 0.2787639
    4.75      0.3911007  - 0.3141686
    5.        0.3120706  - 0.3132178
    5.25      0.2363239  - 0.2901899
    5.5       0.1677952  - 0.2572100
    5.75      0.1078092  - 0.2230526
    6.        0.0558665  - 0.1936491
    6.25      0.0102790  - 0.1726452
    6.5     - 0.0312944  - 0.1616342
    6.75    - 0.0713415  - 0.1602004
    7.      - 0.1120069  - 0.1660669
    7.25    - 0.1546814  - 0.1755937
    7.5     - 0.1997620  - 0.1846352
    7.75    - 0.2466461  - 0.1894800
    8.      - 0.2939305  - 0.1875105
    8.25    - 0.3397195  - 0.1773972
    8.5     - 0.3819292  - 0.1589106
    8.75    - 0.4185188  - 0.1325642
    9.      - 0.4476323  - 0.0992748
    9.25    - 0.4676678  - 0.0601262
    9.5     - 0.4773013  - 0.0162483
    9.75    - 0.4755040    0.0309496
    10.     - 0.4619349    0.0768392
    10.25   - 0.4376947    0.1155002
    10.5    - 0.4050750    0.1435889
    10.75   - 0.3668033    0.1609207
    11.     - 0.3253325    0.1696744
    11.25   - 0.2824083    0.1731202
    11.5    - 0.2389400    0.1745300
    11.75   - 0.1950957    0.1764743
    12.     - 0.1505291    0.1804641
    12.25   - 0.1046604    0.1868576
    12.5    - 0.0569533    0.1949904
    12.75   - 0.0071341    0.2034867
    13.       0.0446803    0.2106703
    13.25     0.0979588    0.2149603
    13.5      0.1518169    0.2151444
    13.75     0.2051265    0.2104894
    14.       0.2566337    0.2007102
    14.25     0.3050586    0.1858619
    14.5      0.3491643    0.1662135
    14.75     0.3877967    0.1421434
    15.       0.4199023    0.1140695
```