HW3Solutions

Chapter 6 exercise 2gh

```c
/* Part (g) */
int funct3(int i)
{
    /* Add semicolon here */
    return i * i;
}
int i;
i = (int)funct3(i);
funct3(i);
i = funct3(8);

/* Part (h) */
double cube(float);
double cube(float number) /* Add the double return type */
{
    return number * number * number;
}
```

Chapter 6 exercise 8d

```c
#include <stdio.h>
#include <math.h>
#include <chplot.h>

#define NUM_POINTS 100

double f4(double x)
{
    double retval;

    retval = (3 * x * x + 4 * x + 3) / (5 * sin(x * x) + 4 * x * x
    + 3);
    return retval;
}

int main()
{
    double x, x0, xf, xstep, result;
    int i, n;

    printf("    x        f4(x)\n");
    printf("  ---------------\n");
    x0 = -1.0;                  /* initial value for x */
    xf = 5.0;                   /* final value for x */
    xstep = 0.25;               /* step size for x */
    n = (xf - x0) / xstep + 1; /* number of points */
    for (i = 0; i < n; i++)
    {
        x = x0 + i * xstep; /* calculate value x */
        result = f4(x);
        printf("%8.4f %8.4f\n", x, result);
    }
```

```
    fplotxy(f4, x0, xf, NUM_POINTS, "function f4(x)", "x", "y");
    return 0;
}
```

Chapter 6 exercise 9d

```
#include <stdio.h>
#include <math.h>
#include <chplot.h>

#define NUM_X_POINTS 100
#define NUM_Y_POINTS 100

double f8(double x, double y)
{
    double retval;

    retval = (3 * x * x + 4 * y + 3) / (5 * sin(y * y) + 4 * x * x + 6);
    return retval;
}

int main()
{
    double x, x0, xf, xstep,
        y, y0, yf, ystep, result;
    int i, j, nx, ny;

    printf("      x           y      f8(x,y)\n");
    printf("  -----------------------------\n");
    x0 = -1.0;
    xf = 5.0;
    xstep = 1.0;
    nx = (xf - x0) / xstep + 1; /* num of points for x */
    y0 = 2.0;
    yf = 4.0;
    ystep = 0.5;
    ny = (yf - y0) / ystep + 1; /* num of points for y */
    result = f8(-0.6970, 2.2020);

    for (i = 0; i < nx; i++)
    {
        x = x0 + i * xstep; /* calculate value for x */
        for (j = 0; j < ny; j++)
        {
            y = y0 + j * ystep; /* calculate value for y */
            result = f8(x, y);
            printf("%10.4f %10.4f %8.4f\n", x, y, result);
        }
    }
    fplotxyz(f8, x0, xf, y0, yf, NUM_X_POINTS, NUM_Y_POINTS, "f8(x,y)", "x", "y", "x");
    return 0;
}
```

Chapter 6 exercise 14

```c
/**
 * Use a function to compute the volume of a sphere.
 * Author: Nicolas Ventura
 */

#include <stdio.h>
#include <math.h>

double volume(double r) {
   return (4.0 / 3.0) * M_PI * (r * r * r);
}

int main(void) {
   double r = 5.0;
   double vol;

   /* Calculate the volume. */
   vol = volume(r);

   /* Display the result. */
   printf("volume = %f m^3\n", vol);

   return 0;
}
```

Chapter 6 exercise 31

```c
/**
 * Plot the humps function.
 */

#include <math.h>
#include <chplot.h>

double humps(double x) {
    /* function to be plotted */
    return 1.0 / ((x - 0.3) * (x - 0.3) + 0.01) + 1 / ((x - 0.9) *
    (x - 0.9) + 0.04) - 6;
}

int main(void) {
    char *title = "Humps function", *xlabel = "x", *ylabel = "y";
    double x0 = -1, xf = 2;
    int num = 200;

    printf("humps(0.3) = %f\n", humps(0.3));
    printf("humps(0.9) = %f\n", humps(0.9));
    fplotxy(humps, x0, xf, num, title, xlabel, ylabel);
    return 0;
}
```

Chapter 6 exercise 32

```c
/**
 * Plot the peaks function.
 */

```

```c
#include <math.h>
#include <chplot.h>

double peaks(double x, double y) {
    // function to be plotted
    return 3 * (1 - x) * (1 - x) * exp(-(x * x) - (y + 1) * (y + 1)
    ) - 10 * (x / 5 - x * x * x - pow(y, 5)) * exp(-x * x - y * y)
    - 1 / 3 * exp(-(x + 1) * (x + 1) - y * y);
}

int main(void) {
    char *title = "Peaks function", *xlabel = "x", *ylabel = "y", *
    zlabel = "z";
    double x0 = -3, xf = 3, y0 = -4, yf = 4;
    int x_num = 60, y_num = 80;

    fplotxyz(peaks, x0, xf, y0, yf, x_num, y_num, title, xlabel,
    ylabel, zlabel);
    printf("peaks(1.5, 2.5) = %f\n", peaks(1.5, 2.5));
    return 0;
}
```

Chapter 6 exercise 35

```c
#include <stdio.h>
#include <math.h>
#include <chplot.h>

double overdamped(double t)
{
  return 4.2 * exp(-1.57 * t) - 0.2 * exp(-54.2 * t);
}

double criticaldamped(double t)
{
  return 4 * (1 - 3 * t) * exp(-3 * t);
}

double underdamped(double t)
{
  return 4 * exp(-0.5 * t) * sin(3 * t + 2);
}

int main()
{
  double t0, tf, t;
  int num = 100; // number of points for plotting
  CPlot plot;

  t = 2;
  printf("Distance x for the overdamped system = %f\n", overdamped(
    t));
  printf("Distance x for the critically damped system = %f\n",
    criticaldamped(t));
  printf("Distance x for the underdamped system = %f\n",
    underdamped(t));

```

```
  t0 = 0;
  tf = 10;
  plot.title("Damped free vibration");
  plot.label(PLOT_AXIS_X, "time (second)");
  plot.label(PLOT_AXIS_Y, "x");
  plot.func2D(t0, tf, num, overdamped);
  plot.legend("overdamped", 0);
  plot.func2D(t0, tf, num, criticaldamped);
  plot.legend("critically damped", 1);
  plot.func2D(t0, tf, num, underdamped);
  plot.legend("underdamped", 2);
  plot.plotting();

  /* Settling Time Calculation:
     * Initial Value: 4
     * Final Value: 0
     * Settled Response Range: 0 +- 4(.02) = -.08 < 0 < .08 */
  /* General Solution Method: Go backwards from a time which we know the
     * system to be settled. As soon as we encounter a time which the system
     * is not in the 2% settled bounds, that is the settling time of the
     * system. */
  /* Overdamped */
  for (t = 20; t > 0; t = t - 0.01)
  {
    if (fabs(overdamped(t)) > 0.08)
    {
      printf("2 percent Ts overdamped: %lf seconds\n", t);
      break;
    }
  }

  /* Critically Damped */
  for (t = 20; t > 0; t = t - 0.01)
  {
    if (fabs(criticaldamped(t)) > 0.08)
    {
      printf("2 percent Ts critically damped: %lf seconds\n", t);
      break;
    }
  }

  /* Under Damped */
  for (t = 20; t > 0; t = t - 0.01)
  {
    if (fabs(underdamped(t)) > 0.08)
    {
      printf("2 percent Ts underdamped: %lf seconds\n", t);
      break;
    }
  }
}

/* text output in a console:
Distance x for the overdamped system = 0.181788
```

```
Distance x for the critically damped system = -0.049575
Distance x for the underdamped system = 1.455858

2 percent Ts overdamped: 2.520000 seconds
2 percent Ts critically damped: 1.790000 seconds
2 percent Ts underdamped: 7.390000 seconds
*/
```

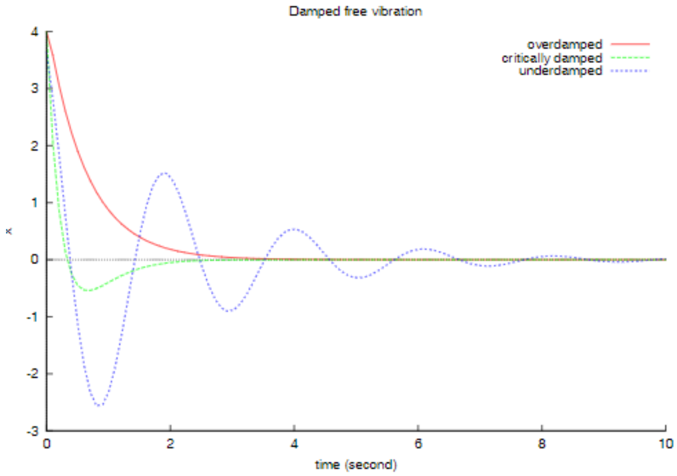

Image 1: Response graph

Chapter 6 exercise 42b

```
/* Find the cubic root of 3 */
#include <stdio.h>
#include <math.h>  /* for fabs() */
#include <float.h> /* for FLT_EPSILON */

#define NUM_POINTS 100 /* number of points for plotting */
#define N 100

double a = 3.0;

/* function for x^3 - 3 */
double func(double x)
{
  double retval;

  retval = x * x * x - a;
  return retval;
}

/* derivative of function */
double funcp(double x)
{
  double retval;

  retval = 3 * x * x;
```

```
  return retval;
}

int main(void)
{
  int i;
  double x0, xf, x1, x2;

  x0 = -5.0; /* initial value for x */
  xf = 5.0;  /* final value for x */

  fplotxy(func, x0, xf, NUM_POINTS, "function f(x)", "x", "f(x)");

  x1 = 2.0;
  for (i = 1; i <= N; i++)
  {
    x2 = x1 - func(x1) / funcp(x1);
    if (fabs(x2 - x1) < FLT_EPSILON)
      break;
    x1 = x2;
  }
  if (i < N)
  {
    printf("x = %f\n", x2);
    printf("func(%f) = %f\n", x2, func(x2));
  }
  else
  {
    printf("Newton's method failed to converge\n");
  }
  return 0;
}
```

Chapter 18 exercise 5

```
/**
 * Calculate the volume and surface area of
 * a sphere and return those values as
 * function inputs.
 * Author: Nicolas Ventura
 */

#include <stdio.h>
#include <math.h>

void sphere(double r, double &volume, double &surface)
{
    volume = (4.0 / 3.0) * M_PI * r * r * r;
    surface = 4.0 * M_PI * r * r;
}

int main(void)
{
    double r, volume, surface;
    printf("Enter the radius (r) for a sphere in meters: ");
    scanf("%lf", &r);
    sphere(r, volume, surface);
```

```
    printf("volume = %lf (m^3)\n", volume);
    printf("surface = %lf (m^2)\n", surface);
    return 0;
}
```