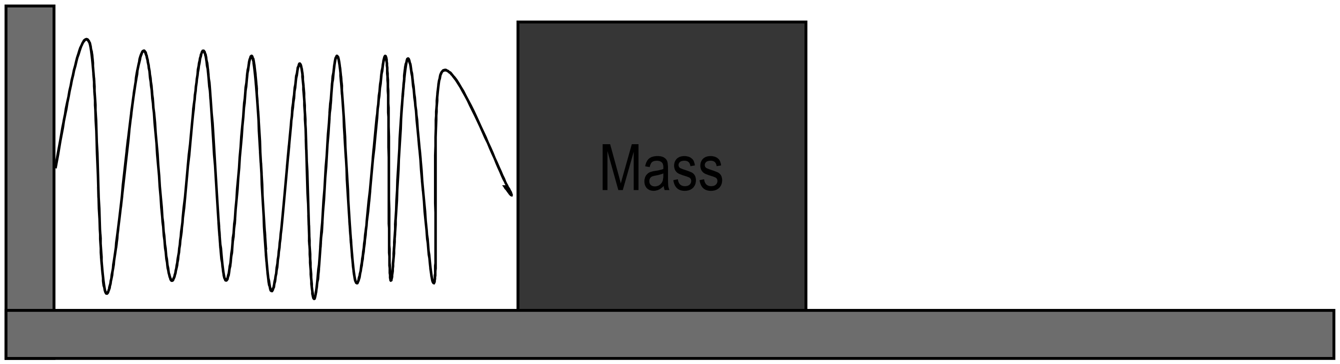
Computers in Engineering COMP 208

Initial Value Problems – Part 2

Michael A. Hawker

Spring-Mass Problem

- Provides us with an example of a system of differential equations in two variables to be solved numerically



The Problem

- Spring-mass systems are basic to courses on the dynamics of vibration
- They are ideal systems composed of a spring and a mass which can oscillate.
- We will simulate such a system with coulomb friction, that is friction which is proportional to the normal force applied and with no damping

The Problem

- The system is characterized by:
  - *mu* the coefficient of kinetic friction between the mass and the surface
  - *m* the mass of the system
  - *k* the stiffness of the spring

The Equations

We describe the motion of the system in the x direction with respect to time by two equations in two variables.

The displacement can be defined as `x(t)`

And the velocity as `y(t) = dx/dt`

The equations become

```
dx/dt = y
dy/dt = -mu*g*sign(y) - k*x/m
```

Initial Value Problems In Two Variables

- This is an example of an initial value problem in two variables.
- In the example, the initial values are the initial displacement of x, say 4.5 and the initial velocity, y=0
- We can use an Euler method or a Runge-Kutta method in two variables to approximate a solution

Euler Method

```
typedef double (*DfDDD)
               (double, double, double);

void Euler2-Step(double t, double h,
                 double *x, double *y,
                 DfDDD xp, DfDDD yp){
  double x_tmp = *x;
  *x  += h * xp(t, *x, *y) ;
  *y  += h * yp(t, x_tmp,*y);
}
```

A Note on Typedef

Without typedef we could still specify type of function that must be passed.

The function heading could have been:

```
void Euler2-Step(double t, double h,
         double *x, double *y,
         double *xp (double, double, double),
         double *yp (double, double, double))
```

Solving the Spring-Mass Problem

```
int main(void)
{
  FILE * outFile = fopen("Stest.csv","w+");
  double x, xp, y, t = 0.0;
  mu = 0.3; m = 1000.0; k = 5000.0;
  x = 4.5; y = 0.0;
  if (outFile)
  {
    do {
      fprintf(outFile, "%g, %g\n", t, x);
      xp = x;
      RK2_Step(t, H, &x, &y, xPrime, yPrime);
      t += H;
    } while (t < 6.0);
  fclose(outFile);
  }
  else printf("Could not open the file");
  return 0;
}
```

Runge-Kutta Review

```
void RK_step(double h, double x, double
  *y,
              DfDD f){
  double k1, k2, k3, k4, half = h/2.0;

  k1 = f(x, *y);
  k2 = f(x + half, *y + half * k1);
  k3 = f(x + half, *y + half * k2);
  k4 = f(x + h, *y + h*k3);

  *y += (h/6.0) * (k1 + 2.0*k2 + 2.0*k3 +
  k4);
}
```

Runge-Kutta in Two Variables

```
void RK2-Step(double t, double h, double *x,
  double *y, DfDDD xp, DfDDD yp){
  double h_half = h/2.0,k1,j1,k2,j2,k3,j3,k4j4;
  k1 = xp(t, *x, *y);
  j1 = yp(t, *x, *y);
  k2 = xp(t+h_half,*x+h_half*k1,*y+h_half*j1);
  j2 = yp(t+h_half,*x+h_half*k1,*y+h_half*j1);
  k3 = xp(t+h_half,*x+h_half*k2,*y+h_half*j2);
  j3 = yp(t+h_half,*x+h_half*k2,*y+h_half*j2);
  k4 = xp(t+h,*x+h*k3,*y+h*j3);
  j4 = yp(t+h,*x+h*k3,*y+h*j3);
  *x += (h/6.0)*(k1+2*k2+2*k3+k4);
  *y += (h/6.0)*(j1+2*j2+2*j3+j4);
}
```

fopen

- "r" Open a file for reading. The file must exist.
- "w" Create an empty file for writing. If a file with the same name already exists its content is erased and the file is treated as a new empty file.
- "a" Append to a file. Writing operations always append data at the end of the file. The file is created if it does not exist.
- "+" opens for reading and writing following above
- "b" opens file as raw byte-for-byte binary data

Global Variables (again)

Some variables are used in different functions.

If they were declared in main, they would not be accessible to other functions

We can declare them globally, that is outside of main and every function in the file can use them

```
double H = 0.01, G = 9.81;
double mu, m, k;
```

Using Global Variables

```
double H = 0.01,    G = 9.81;
double mu, m, k;
int main(void){
  mu = 0.3; m = 1000.0; k = 5000.0;
  . . .
      RK2_Step(t, H, &x, &y, xPrime, yPrime);
  . . .
}
double xPrime(double t, double x, double y){
  return y;
}
double yPrime(double t, double x, double y){
  return -x * k / m - mu * G * sign(y);
}
```

In relation to #define

- Global Variables can be modified
- While this is useful, if gone unchecked can result in hard to find errors
- #define is a find and replace
- No extra memory is used

Function Prototype – Review

A function prototype provides enough information to enable the compiler to make sure the function is used properly

It must appear before the function is called.

In allows the compiler to determine

1. the number of parameters
2. the types of the parameters
3. the type of value returned

Function Prototype – Review

The prototype has the form:

```
return_type function_name
    (type of p1, type of p2, ...,
     type of pn)
```

The body of the function is not needed until the function is actually used after it is compiled

Parameter names are allowed but are not needed

Prototyping in Spring-Mass

```
#include <stdio.h>
#include <math.h>
typedef double (*DfDDD) (double, double, double);
double xPrime(double, double, double);
double yPrime(double, double, double);
int sign(double);
void RK2_Step(double, double, double *, double *,
              func, func);
double H = 0.01,    G = 9.81;
double mu, m, k;
int main(void){
  . . .
   RK2_Step(t, H, &x, &y, xPrime, yPrime);
  . . .
}
```

Alternate Prototypes

Since parameter names are only for reference purposes, if we want we could include them

```
double xPrime(double t, double x, double y);
double yPrime(double t, double x, double y);
int sign(double x);
void RK2_Step(double t, double h,
              double *x, double *y,
              DfDDD xp, DfDDD yp);
```

Mutual Recursion

Prototyping is necessary when functions call one another.

```
int odd (int a){
  if (a == 0) return 0;
  else return even(a-1);
  }

int even (int a){
  if (a == 0) return 1;
  else return odd(a-1);
  }
```

Even-Odd Prototyping

```
int odd(int);
int even(int);
int main (){
  int x=22;
  if (even(x))
     printf (" %d is even\n", x);
  else
     printf(" %d is odd\n", x);
  return 0;
}
```

Spring-Mass Solution (a)

```
#include <stdio.h>
#include <math.h>
#define HSTEP 0.01
#define GRAV 9.81
#define MU 0.3
#define MASS 1000.0
#define KVAR 5000.0
typedef double (*func) (double, double, double);

double xPrime(double, double, double);
double yPrime(double, double, double);
int sign(double);
void RK2_Step(double, double, double *, double *,
              func, func);
```

Spring-Mass Solution (b)

```
int main(void){
  FILE * outFile = fopen("Stest.csv","w+");
  double x, xp, y, t = 0.0;
  x = 4.5; y = 0.0;
  if (outFile){
    do {
      fprintf(outFile, "%g, %g\n", t, x);
      xp = x;
      RK2_Step(t, HSTEP, &x, &y, xPrime, yPrime);
      t += HSTEP;
    } while (t < 6.0);
  fclose(outFile);
  }
  else printf("Could not open the file");
  return 0;
}
```

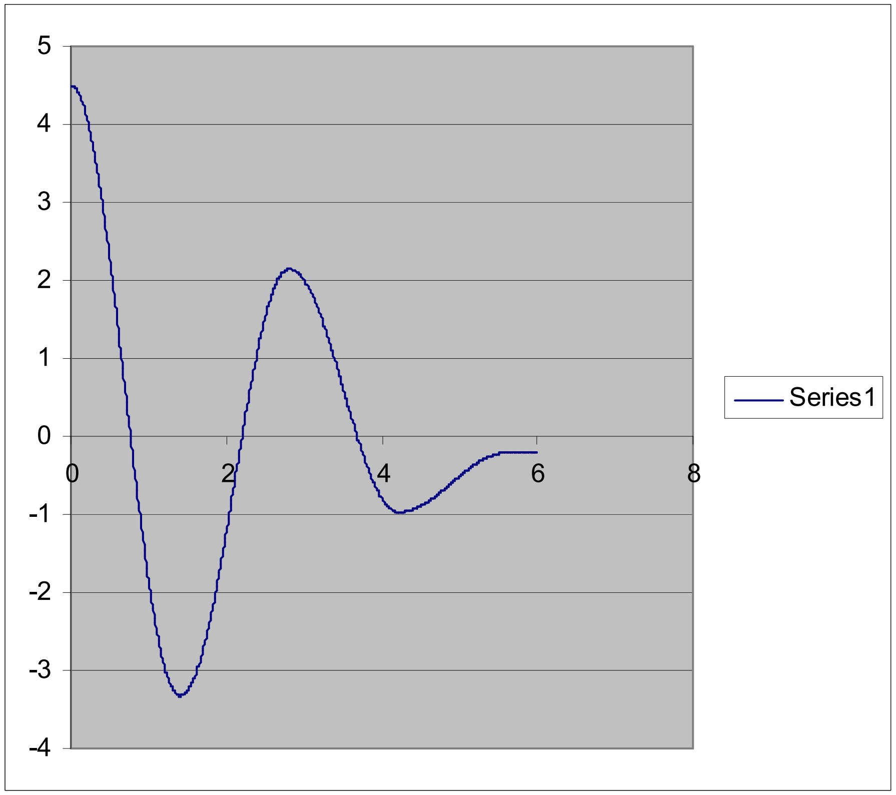
Spring-Mass Solution (c)

```
double xPrime(double t, double x, double y){
  return y;
}

double yPrime(double t, double x, double y){
  return -MU * GRAV * sign(y) -x * KVAR / MASS;
}

int sign(double x){
  return (x == 0) ? 0 : ((x < 0.0) ? -1 : 1);
}
```

Spring-Mass Solution (d)

```
void RK2_Step(double t, double h, double *x, double *y,
              DfDDD xp, DfDDD yp){
  double h_half = h/2.0,k1,j1,k2,j2,k3,j3,k4,j4;
  k1 = xp(t, *x, *y);
  j1 = yp(t, *x, *y);
  k2 = xp(t+h_half,*x+h_half*k1,*y+h_half*j1);
  j2 = yp(t+h_half,*x+h_half*k1,*y+h_half*j1);
  k3 = xp(t+h_half,*x+h_half*k2,*y+h_half*j2);
  j3 = yp(t+h_half,*x+h_half*k2,*y+h_half*j2);
  k4 = xp(t + h, *x + h * k3, *y + h * j3);
  j4 = yp(t + h, *x + h * k3, *y + h * j3);
  *x += (h / 6.0) * (k1 + 2 * k2 + 2 * k3 + k4);
  *y += (h / 6.0) * (j1 + 2 * j2 + 2 * j3 + j4);
}
```

Excel Generated Graph


Series1