McGill

COMP 208 – Computers in Engineering

COMP 208
Computers in Engineering

Lecture 21

Jun Wang
School of Computer Science
McGill University
Fall 2007

Review

- Numerical root-finding methods:
  - bisection
  - secant
  - false position: a variation of bisection
  - Newton-Raphson (tangent line)

- Idea: generating a sequence of numbers that converge to the root
  - termination condition:
    - $|f(x_n)| <$ tolerance
    - $|x_n - x_{n-1}| <$ tolerance

Review

- Numerical differentiation
  - 2 point approximation
  - 3 point approximation

```
double centered3_diff(DfD f, double x, double h)
{
   return (f(x + h) - f(x - h)) / (2 * h);
}
```

- Initial value problem (ODE)
  - Idea: the ODE is hard to solve analytically, so we solve it numerically by generating the value of the function y(x) for a list of x values
  - Euler method
    - given $y'(x) = f(x, y)$, $y(x_0) = y_0$
    - then for $x_{i+1} = x_i + h$,
      we have $y_{i+1} = h*y'(x_i) + y_i \approx h*f(x_i, y_i) + y(x_i)$

Runge-Kutta Method

The Euler method is not very accurate since the error tends to keep growing.

In the (fourth-order) Runge_Kutta method the derivative is evaluated four times

1. At the initial point
2. Twice at a trial midpoint
3. At a trial endpoint

Runge-Kutta Formula

Use the following to compute the next step

$$k1 = f(x_n, y_n)$$

$$k2 = f(x_n+h/2, y_n + k1/2)$$

$$k3 = f(x_n+h/2, y_n + k2/2)$$

$$k4 = f(x_n+h, y_n + k3)$$

$$y_{n+1} = y_n + (k1 + 2*k2 + 2*k3 + k4)*h/6$$

Implementing Runge-Kutta

```
void RK_step(double h, double x, double *y,
             fun 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);

}
```

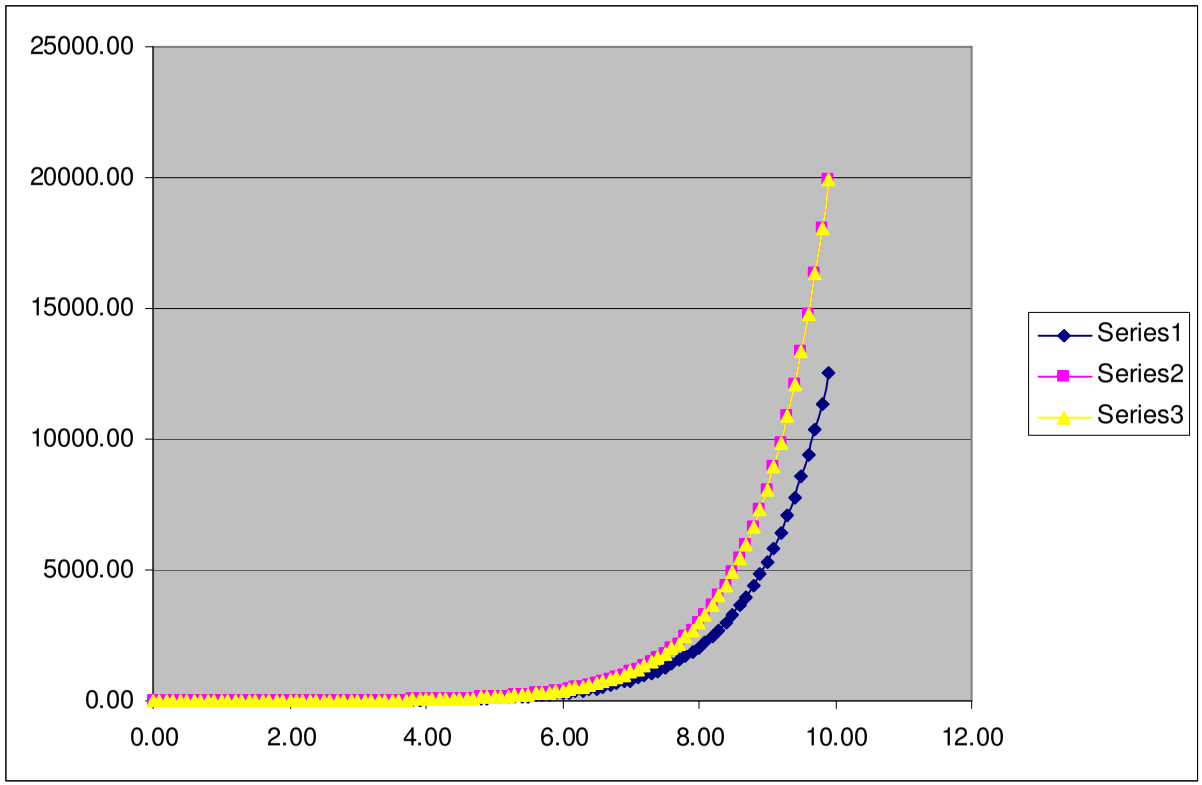
Comparing Runge-Kutta and Euler

```
int main(){
   double a, b, x, y
   FILE * myfile = fopen("RKtest.csv","w+");
   int i,steps = 100;
   if (myfile){
      x = 0; a = 0;
      y = 1; b = 1;
      for (i = 0; i <= steps; i++){
         fprintf (myfile," %f, %f, %f, %f\n", x, y, b, exp(x)-x);
         Euler_step(H,x,&y,func);
         RK_step(H,a,&b,func);
         x += H;
         a += H;
      }
      fclose(myfile);
   }
   else printf("Could not open the file");
   return 0;
}
```

Part of the Output File

x	Euler	RK	exp(x)-x
3.700000,	30.303949,	36.747190,	36.747304
3.800000,	33.604343,	40.901054,	40.901184
3.900000,	37.244778,	45.502301,	45.502449
4.000000,	41.259256,	50.597983,	50.598150
4.100000,	45.685181,	56.240098,	56.240288
4.200000,	50.563699,	62.486116,	62.486331
4.300000,	55.940069,	69.399551,	69.399794
4.400000,	61.864076,	77.050594,	77.050869
4.500000,	68.390484,	85.516821,	85.517131
4.600000,	75.579532,	94.883965,	94.884316
4.700000,	83.497485,	105.246776,	105.247172
4.800000,	92.217234,	116.709970,	116.710418
4.900000,	101.818957,	129.389275,	129.389780
5.000000,	112.390853,	143.412590,	143.413159
5.100000,	124.029938,	158.921266,	158.921907
5.200000,	136.842932,	176.071519,	176.072242
5.300000,	150.947225,	195.035996,	195.036810
5.400000,	166.471948,	216.005499,	216.006416
5.500000,	183.559142,	239.190900,	239.191932
5.600000,	202.365057,	264.825246,	264.826407
5.700000,	223.061562,	293.166095,	293.167401
5.800000,	245.837719,	324.498091,	324.499560
5.900000,	270.901490,	359.135816,	359.137468
6.000000,	298.481640,	397.426937,	397.428793
6.100000,	328.829803,	439.755685,	439.757770
6.200000,	362.222784,	486.546699,	486.549041

Excel Generated Graph



McGill

COMP 208 – Computers in Engineering

IVP (Part 2)

Nathan Friedman

Spring-Mass Problem

This problem was prepared by Jean Francois Bastien as an assignment for this course in a previous year

It 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) - kx/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 ($t_0 = 0$, $x_0 = 4.5$, $y_0 = 0$)

We can use an Euler method or a Runge-Kutta method in two variables to approximate a solution

Initial Value Problems In Two Variables

Given:

$x'(t) = F(t, x, y)$

$y'(t) = G(t, x, y)$

$x_0 = x(t_0)$, $y_0 = y(t_0)$

Approximate,

$x_i \approx x(t_i)$, and $y_i \approx y(t_i)$, where $t_i = i*h + t_0$

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);
}
```

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

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))
```

Runge-Kutta in Two Variables

```
void RK2-Step(double t, double h, double *x, double *y,
   func xp, func 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);
}
```

Solving the Spring-Mass Problem

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

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;
}
```

This is not a complete program!

Global Variables

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);
}
```

Function Prototype – A 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.

It allows the compiler to determine

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

Function Prototype – A 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 (*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);
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,
              func xp, func yp);
```

Spring-Mass Solution (a)

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

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

Spring-Mass Solution (b)

```
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;
}
```

Spring-Mass Solution (c)

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

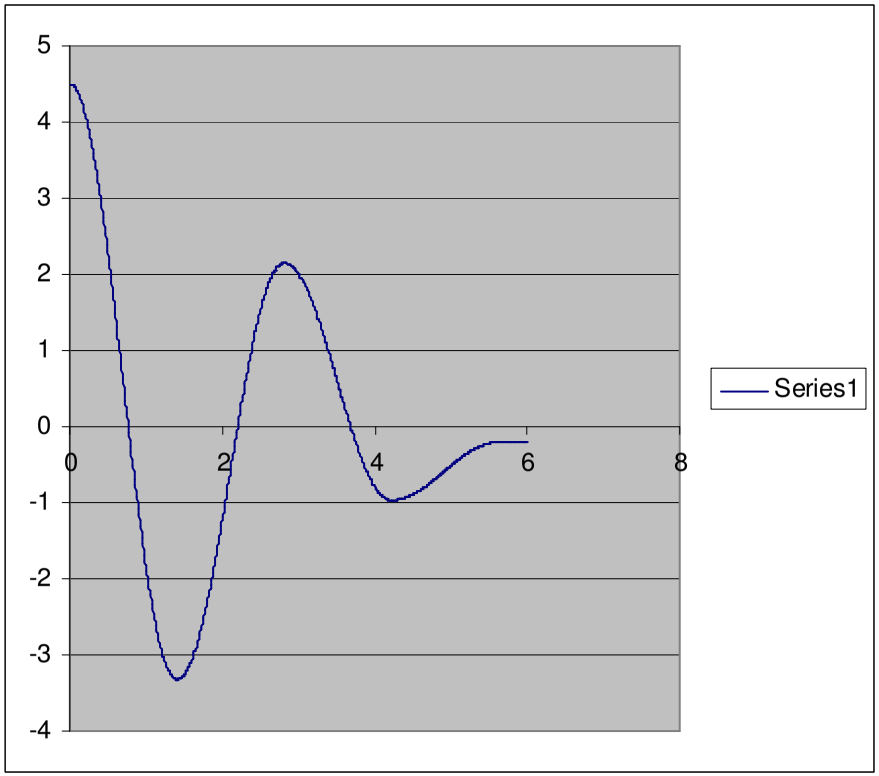
Problematic! When compare 2 decimal numbers a, b, for equality, always use |a-b| < tolerance

demo

Spring-Mass Solution (d)

```
void RK2_Step(double t, double h, double *x, double *y,
              func xp, func 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



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;
}
```

McGill

COMP 208 – Computers in Engineering

Numerical Integration

Nathan Friedman

Integration

- Many applications require evaluating the integral of a function
- The integrals of many elementary functions cannot be derived analytically
- As we have seen, we may not even have an analytic form for the function. We may just be able to sample it at various points

Integration

- This lead to the development of techniques for evaluating such integrals numerically
- Numerical integration techniques predate the use of electronic computers