# 16.070 Introduction to Computers and Programming

**March 21** **Recitation 7** **Spring 2002**

## *Topics:*

- Quick review of PS5 issues
- Concept
- Equations of Motion
- Continuous State Space
- Discrete State Space
- Algorithm
- Coding
- Analysis

## Problem Set Issues

A few more administrative issues have come up in the past week.

1. There is a very specific convention for filenames at the end of each homework problem that should be used. If the conventions are not followed we cannot guarantee that your homework will be identified and graded.

2. There have been complaints that students in the labs are not cleaning up after themselves. Follow the rule of camping: "Leave the place cleaner than you found it"

3. The Handyboard does not have file i/o functions.

## Procedure for the Simulation of Differential Equations

An Earth Lander(/rock/ball) is suspended 500m from the surface of the earth and released. Simulate the dynamics of the problem as the vehicle falls to earth. Only 1-DOF is required: the vertical altitude of the vehicle above the surface of the earth. Assume a constant gravitational acceleration of g=9.81m/s/s. The vehicles mass is 500kg (does this matter?). The vehicle does not thrust and drag is neglected.
Write the vehicles state history to a file, in the format: `<time> <displacement> <velocity>`

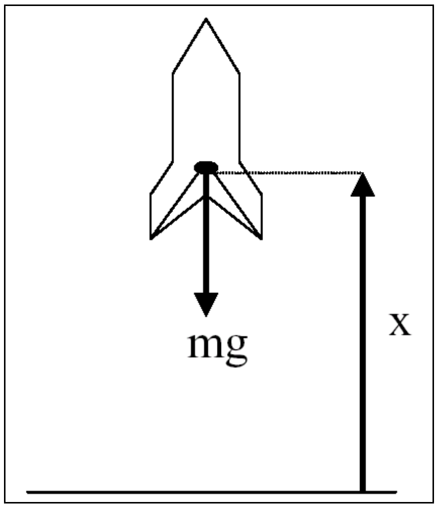


Figure 1 – Lander suspended above the surface of the Earth (NO THRUST!)

## Equations of motion

Newton's second law: $F = ma = m\ddot{x}$

Initial Conditions: $x(0) = 500m$

$\dot{x}(0) = 0\,\tfrac{m}{s}$

Forces acting on the vehicle: $F = -mg$

Complete simplified equation of motion: $m\ddot{x} = -mg$

## Continuous State Space

Using the iterative Euler approach, an initial condition is required for each state:

State Vector: $X = \begin{Bmatrix} x_1 \\ x_2 \end{Bmatrix} = \begin{Bmatrix} x \\ \dot{x} \end{Bmatrix}$

Continuous State Space Representation: $\dot{X} = \begin{bmatrix} 0 & 1 \\ 0 & 0 \end{bmatrix} X + \begin{bmatrix} 0 \\ \tfrac{1}{m} \end{bmatrix} (-mg)$

## Discrete State Space

Approximate $\dfrac{dx}{dt}$ with $\dfrac{x_{n+1} - x_n}{\Delta t}$ . Then we get the Discrete State Space result:

$$\begin{Bmatrix} \dfrac{x_{1_{n+1}} - x_{1_n}}{\Delta t} \\ \dfrac{x_{2_{n+1}} - x_{2_n}}{\Delta t} \end{Bmatrix} = \begin{bmatrix} 0 & 1 \\ 0 & 0 \end{bmatrix} \begin{Bmatrix} x_{1_n} \\ x_{2_n} \end{Bmatrix} + \begin{bmatrix} 0 \\ \dfrac{1}{m} \end{bmatrix} (-mg)$$

$$\begin{Bmatrix} x_{1_{n+1}} - x_{1_n} \\ x_{2_{n+1}} - x_{2_n} \end{Bmatrix} = \begin{bmatrix} 0 & \Delta t \\ 0 & 0 \end{bmatrix} \begin{Bmatrix} x_{1_n} \\ x_{2_n} \end{Bmatrix} + \begin{bmatrix} 0 \\ \dfrac{\Delta t}{m} \end{bmatrix} (-mg)$$

$$\begin{Bmatrix} x_{1_{n+1}} \\ x_{2_{n+1}} \end{Bmatrix} = \begin{bmatrix} 1 & \Delta t \\ 0 & 1 \end{bmatrix} \begin{Bmatrix} x_{1_n} \\ x_{2_n} \end{Bmatrix} + \begin{bmatrix} 0 \\ \dfrac{\Delta t}{m} \end{bmatrix} (-mg)$$

## Algorithm

**Matrices:**

$$A = \begin{bmatrix} 1 & \Delta t \\ 0 & 1 \end{bmatrix}$$

$$B = \begin{bmatrix} 0 \\ \dfrac{\Delta t}{m} \end{bmatrix}$$

$$t_0 = 0$$

**Variables and constants:**

$$\Delta t = 0.01s$$

$$t_{\max} = 20s$$

$$X_n$$

$$X_{n+1}$$

$$F = -mg$$

**Basic Pseudo-code**

MAIN MODULE

```
1.  START
2.  Declare all the variables/constants above
3.  Open an output file for telemetry
4.  Set initial conditions
5.  Write initial states to a file
6.  WHILE (t <= $t_{max}$)
7.          Increment time (t = t+dt)
8.          Update State ( $X_{n+1} = AX_n + BF$ )
9.          Write states to file
10.         Advance the state: $X_n = X_{n+1}$
11. END WHILE
12. Close all open files
13. EXIT
```

## Code

```
/* -------------------------------- */
/* Thomas Jones, April 2001         */
/* Recitation 7  (2002)             */
/* Simulation of a falling lander   */
/* (on earth)                       */
/* -------------------------------- */
/* ALL UNITS ARE STANDARD METRIC */
#define TIME_STEP 0.01
#define OUT_FILE "rec7_out.txt"
#define MASS 500.0
#define GRAV_ACCEL 9.81
#define INITIAL_POSITION 500.0
#define INITIAL_VELOCITY 0.0
#define MAX_TIME 10.0

#include <stdio.h>
#include <stdlib.h>

/* Prototypes */
void Set_Initial_Conditions(double Xn[]);
void Write_To_File(double t, double X[],FILE *ofp);
void Update_X(double A[][2],double B[],double F,double Xn[],double Xnp1[]);

/* Start main function */
int main(void)  {
  /* Variable Declarations and Setup                   */
  /* declare time, MASS and gravitational acceleration */
  double t=0.0;               /* time                  */
  const double dt=TIME_STEP;  /* time step             */
  const double m=MASS;        /* MASS                  */
  const double g=GRAV_ACCEL;  /* gravity acceleration  */

  /* declare A, B, F, Xn, Xnp1, such that Xnp1 = AXn + BF */
  double A[2][2]={1.0, dt, 0.0, 1.0};
  double B[2]={0.0, dt/m};
  double Xn[2]={0.0, 0.0};
  double Xnp1[2]={0.0,0.0};
  double F=-m*g;

  /* open telemetry output file */
  FILE *ofp;
  printf("Opening Telemetry File...\n");
  if ((ofp=fopen(OUT_FILE,"w+"))==0) {
    printf("File open error!\n");
    exit(-1);
  }
  printf("File Opened...\n");

  /* Implementation of Pseudocode                      */
  /* Main BODY of Code                                 */
  /* --------------------------------------------      */
  printf("Initializing State Variables...\n");
  Set_Initial_Conditions(Xn);   /* initial conditions    */

  Write_To_File(t,Xn,ofp);      /* save inital telemetry */

  printf("Simulating...Please wait...\n");
  while(t <= MAX_TIME) {
    t=t+dt;                     /* increment time */
    Update_X(A,B,F,Xn,Xnp1);    /* state update   */
    Write_To_File(t,Xnp1,ofp);  /* save telemetry */
    Xn[0]=Xnp1[0];              /* advance state  */
    Xn[1]=Xnp1[1];
  }
  printf("Simulation Complete!\n");
  /* ---------------------- */

  /* close all open files */
  printf("Closing Files...\n");
  fclose(ofp);
  printf("Done!\n\n");
  return 0;
} /* end main */
```

```
/* ----------------------------------------- */
/* Function sets initial conditions of state */
/* ----------------------------------------- */
void Set_Initial_Conditions(double Xn[])
{
  Xn[0]=INITIAL_POSITION;
  Xn[1]=INITIAL_VELOCITY;
  return;
} /* end Set_Initial_Conditions */


/* ------------------------------------------ */
/* Function writes state and time to file ofp */
/* ------------------------------------------ */
void Write_To_File(double t, double X[],FILE *ofp)
{
  fprintf(ofp,"%2.2lf\t%lf\t%lf\n",t,X[0],X[1]);
  return;
} /* end Write_To_File */


/* ------------------------------------------ */
/* Function performs state update */
/* ------------------------------------------ */

/* PS7 requires this state update to be performed by    */
/* a general matrix multiplier function, which is quite */
/* different from the code below !!!                    */

void Update_X(double A[][2],double B[],double F,double Xn[],double Xnp1[])
{
  Xnp1[0]=A[0][0]*Xn[0]+A[0][1]*Xn[1]+B[0]*F;
  Xnp1[1]=A[1][0]*Xn[0]+A[1][1]*Xn[1]+B[1]*F;
  return;
} /* end Update_X */
```

## PS7 Specific Issues

In PS7, you will be designing and programming a satellite docking simulator. The PS7 simulator will differ from the example in this recitation insofar as gravity will not be modeled, but thrust will. Also, the PS7 simulator will be in two dimensions, so larger A and B matrices will be required.

An issue that may arise is the equality of floats. If a floating point value is tested for equality to another number, it is highly unlikely to be true. For instance:

```
    float f = 1.0f;
    int Test = (f==1.0f);
```

could easily result in a value of 0 (false) for the variable Test. The reason is that floating point variables (this includes doubles and floats) are represented using a mantissa/exponent form:

$$\text{mantissa} * 2^{\text{exponent}}$$

where the mantissa and the exponent each have a preallocated number of bits. If the number you are testing for cannot be precisely represented using this form or a precision error was introduced at that any point in your program (this happens all the time with floating point numbers) your variable will contain a value close to the expected value, but not exact.

When working with floats, instead of testing for equality, test for inequality within a certain range (a function would be good for this).

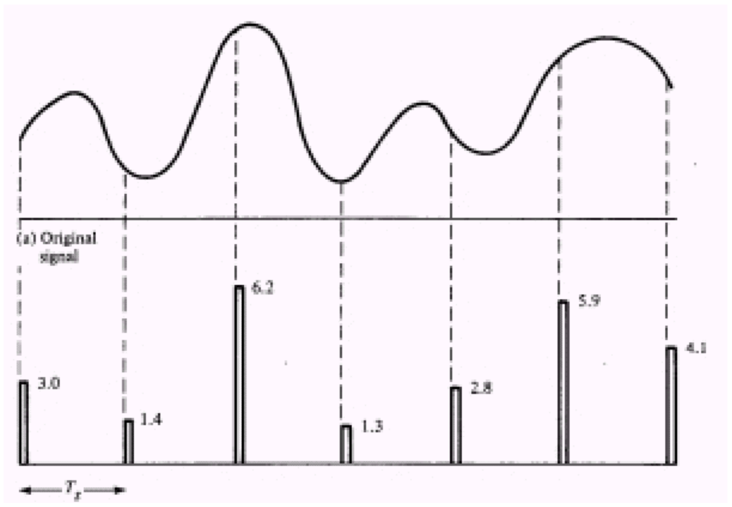
For instance:

```
    float Tolerance = 0.005f;
    float f = 1.0f;
    Int Test = (f>1.0f-Tolerance) && (f<1.0f+Tolerance);
```

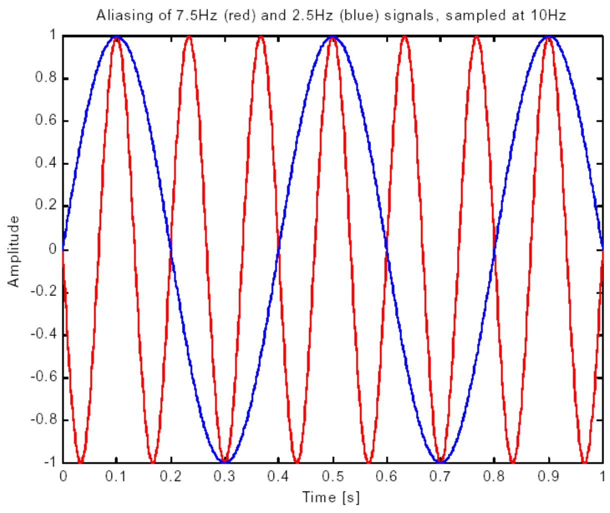
## Sampling

The process of converting naturally occurring measurements in the real world into values in a computer that may be analyzed or acted upon is the process of converting continuous analog data into digital data. Analog data has many advantages in that is easy for humans to process and occurs naturally with virtually infinite detail. Conversely, digital data has the advantage of being easy for computers to process, perfectly reproducible, and quantitative. Examples of analog data are photographs, cassettes, and human memory. Examples of digital data are compact discs, floppy disks, and DSL lines.

In the picture below, data is converted from a continuous curve to a series of discrete point measurements. This is done through a process known as *sampling*, whereby data is measured at a regular interval.



One of the pitfalls of sampling is encountered when the frequency of the continuous data being measured is higher than that of the sampling itself. In the picture below, two separate frequencies are present, but if sampling only occurred every 0.1 seconds, a person trying to analyze the data might fall under the mistaken impression that only 1 frequency was present. This problem is known as aliasing and can be prevented by sampling at more than twice the highest frequency of interest (known as the Nyquist Criterion) and filtering out all frequencies higher than that.


Aliasing of 7.5Hz (red) and 2.5Hz (blue) signals, sampled at 10Hz
Amplitude
Time [s]