MA/CS 375

Fall 2002

Lecture 12
Office Hours

• My office hours
  – Rm 435, Humanities, Tuesdays from 1:30pm to 3:00pm
  – Rm 435, Humanities, Thursdays from 1:30pm to 3:00pm

• Tom Hunt is the TA for this class. His lab hours are now as follow

  – SCSI 1004, Tuesdays from 3:30 until 4:45
  – SCSI 1004, Wednesdays from 12:00 until 12:50
  – Hum 346 on Wednesdays from 2:30-3:30
This Lecture

Solving Ordinary Differential Equations (ODE’s)

• Accuracy
• Stability

• Forward Euler time integrator
• Runge Kutta time integrators

• Newton’s Equations
Ordinary Differential Equation

- Example:

\[
\begin{align*}
    u(0) &= a \\
    \frac{du}{dt} &= -u \\
    u(T) &= ?
\end{align*}
\]

- t is a variable for time
- u is a function dependent on t
- given u at t = 0
- given that for all t the slope of us is –u
- what is the value of u at t=T
Ordinary Differential Equation

- Example:

\[
\begin{align*}
  u(0) &= a \\
  \frac{du}{dt} &= -u \\
  u(T) &= ?
\end{align*}
\]

- we should know from intro calculus that:

\[ u(t) = ae^{-t} \]

- then obviously:

\[ u(T) = ae^{-T} \]
Just in Case You Forgot How…

\[
\frac{du}{dt} = -u \\
\frac{1}{u} \frac{du}{dt} = -1 \quad \text{ok if } u \neq 0 \\
\frac{d \ln(u)}{dt} = -1 \quad \text{ok if } u > 0 \\
\int_{0}^{T} \frac{d \ln(u)}{dt} dt = \int_{0}^{T} -dt \quad \text{integrate in time} \\
\left[ \ln(u) \right]_{t=0}^{T} = [-t]_{0}^{T} \quad \text{Fundamental theorem of calculus} \\
\ln(u(T)) - \ln(u(0)) = -(T - 0) \\
u(T) = u(0)e^{-T}
\]
Family of Solutions

\[ u(t) = u(0)e^{-t} \]

\[ u(0) = 4 \text{ curve} \]

\[ u(0) = -4 \text{ curve} \]
Forward Euler Numerical Scheme

• There are many ways to figure this out on the computer.
• Simplest first.
• We discretize the derivative by

\[
\frac{du(t)}{dt} \approx \frac{u(t + \Delta t) - u(t)}{\Delta t}
\]
Forward Euler Numerical Scheme

- Numerical scheme:

\[
\frac{u^{n+1} - u^n}{\Delta t} = -u^n
\]

- Discrete scheme:

\[
u^{n+1} = (1 - \Delta t)u^n
\]

where: \( u^n \) = approximate solution at \( t = n\Delta t \)
Stability of Forward Euler Numerical Scheme

• Discrete scheme:

\[ u^{n+1} = (1 - \Delta t) u^n \]

• The solution at the n’th time step is then:

\[ u^n = (1 - \Delta t)^n u^0 \]

\[ = (1 - \Delta t)^n u(0) \]
Stability of Forward Euler Numerical Scheme

• The solution at the n’th time step is then:

\[ u^n = (1 - \Delta t)^n u^0 = (1 - \Delta t)^n u(0) \]

• Notice that if \( |1 - \Delta t| > 1 \) then \( |u^n| \) is going to get very large very quickly !!. This is clearly not what we want for an approximate solution to an exponentially decaying exact solution.
Stable Approximations

- $0 < dt < 1$
Stable But Oscillatory Approximations

- $1 \leq dt < 2$
- $dt = 1.25$
- $dt = 1.5$
Unstable (i.e. Bad) Approximations

- $2 < dt$

![Graph showing Unstable Approximations with $dt=4.5$, $dt=3$, and $dt=2.5$]
Summary of dt Stability

- $0 < dt < 1$ stable and convergent since as $dt \to 0$ the solution approached the actual solution.

- $1 \leq dt < 2$ bounded but not cool.

- $2 \leq dt$ exponentially growing, unstable and definitely not cool.
Accuracy of the Forward Euler Scheme

• Next lecture
Application: Newtonian Motion
Two of Newton’s Law of Motions

1) In the absence of forces, an object ("body") at rest will stay at rest, and a body moving at a constant velocity in straight line continues doing so indefinitely.

2) When a force is applied to an object, it accelerates. The acceleration $a$ is in the direction of the force and proportional to its strength, and is also inversely proportional to the mass ($m$) being moved. In suitable units:

$$F = ma$$

with both $F$ and $a$ vectors in the same direction (denoted here in bold face).
Newton’s Law of Gravitation

Gravitational force: an attractive force that exists between all objects with mass; an object with mass attracts another object with mass; the magnitude of the force is directly proportional to the masses of the two objects and inversely proportional to the square of the distance between the two objects.
Real Application

• You can blame Newton for this:

Consider an object with mass $m$

$\begin{align*}
  t &= \text{time} \\
  m &= \text{mass of object} \\
  F &= \text{force on object} \\
  a &= \text{acceleration object} \\
  x &= \text{location of object} \\
  v &= \text{velocity of object}
\end{align*}$

$\begin{align*}
  F &= ma \\
  \frac{d^2x}{dt^2} &= a \\
  \frac{dx}{dt} &= v \\
  \frac{dv}{dt} &= a
\end{align*}$
Two Gravitating Particle Masses

Each particle has a scalar mass quantity
Particle Positions

Each particle has a vector position

\( \mathbf{x}_1 \)

\( \mathbf{x}_2 \)

(0,0)
Particle Velocities

Each particle has a vector velocity

\[ \mathbf{v}_1 \]

\[ \mathbf{v}_2 \]
Particle Accelerations

Each particle has a vector acceleration $\mathbf{a}_1$ and $\mathbf{a}_2$. 
Definition of $\| \cdot \|_2$

- In the following we will use the following notation:

$$\| \mathbf{x} \|_2 = \sqrt{x_1^2 + x_2^2 + x_3^2}$$

- Formally the function $\| \mathbf{x} \|_2$ known as the Euclidean norm of $\mathbf{x}$. It returns the length of the vector $\mathbf{x}$.
Two-body Newtonian Gravitation

- Two objects of mass $M_1$ and $M_2$ exert a gravitational force on each other:

$$F_{12} = \frac{(x_2 - x_1) M_1 M_2 G}{\|x_2 - x_1\|_2^3}$$

$$F_{21} = \frac{(x_1 - x_2) M_2 M_1 G}{\|x_1 - x_2\|_2^3}$$

where $G$ is the gravitational constant.
Newtonian Gravitation

- Newton’s second law (rate of change of momentum = force on body):

\[
\frac{d^2 (M_1 x_1)}{dt^2} = \frac{(x_2 - x_1) M_1 M_2 G}{\|x_2 - x_1\|_2^3}
\]

\[
\frac{d^2 (M_2 x_2)}{dt^2} = \frac{(x_1 - x_2) M_2 M_1 G}{\|x_1 - x_2\|_2^3}
\]
Newtonian Gravitation

- Acceleration:

\[
\frac{d^2 \mathbf{x}_1}{dt^2} = \frac{(\mathbf{x}_2 - \mathbf{x}_1) M_2 G}{\|\mathbf{x}_2 - \mathbf{x}_1\|_2^3}
\]

\[
\frac{d^2 \mathbf{x}_2}{dt^2} = \frac{(\mathbf{x}_1 - \mathbf{x}_2) M_1 G}{\|\mathbf{x}_1 - \mathbf{x}_2\|_2^3}
\]
Newtonian Gravitation

• Using velocity:

\[
\frac{dx_1}{dt} = v_1
\]

\[
\frac{dx_2}{dt} = v_2
\]

\[
\frac{dv_1}{dt} = \frac{\left(\mathbf{x}_2 - \mathbf{x}_1\right) M_2 G}{\left\| \mathbf{x}_2 - \mathbf{x}_1 \right\|_2^3}
\]

\[
\frac{dv_2}{dt} = \frac{\left(\mathbf{x}_1 - \mathbf{x}_2\right) M_1 G}{\left\| \mathbf{x}_1 - \mathbf{x}_2 \right\|_2^3}
\]
N-Body Newtonian Gravitation

• For particle \( n \) out of \( N \)

\[
\frac{d\mathbf{x}_n}{dt} = \mathbf{v}_n
\]

\[
\frac{d\mathbf{v}_n}{dt} = \sum_{i=1,i\neq n}^{i=N} \frac{(\mathbf{x}_i - \mathbf{x}_n) M_i G}{\|\mathbf{x}_i - \mathbf{x}_n\|_2^3}
\]

The force on each particle is a sum of the gravitational force between each other particle
N-Body Newtonian Gravitation Simulation

• Goal: to find out where all the objects are after a time T

• We need to specify the initial velocity and positions of the objects.

• Next we need a numerical scheme to advance the equations in time.

• Can use forward Euler… as a first approach.
Numerical Scheme

For $m=1$ to FinalTime/$dt$
   For $n=1$ to number of objects
      $\mathbf{v}_{n}^{m+1} = \mathbf{v}_{n}^{m} + dt \sum_{i=1,i\neq n}^{i=N} \frac{(\mathbf{x}_{i}^{m} - \mathbf{x}_{n}^{m}) M_{i} G}{\|\mathbf{x}_{i}^{m} - \mathbf{x}_{n}^{m}\|_{2}^{3}}$
   End
   For $n=1$ to number of objects
      $\mathbf{x}_{n}^{m+1} = \mathbf{x}_{n}^{m} + dt \left( \mathbf{v}_{n}^{m} \right)$
   End
End
End
planets1.m Matlab script

• I have written a *planets1.m* script.
• The quantities in the file are in units of
  – kg (kilograms -- mass)
  – m (meters -- length)
  – s (seconds -- time)
• It evolves the planet positions in time according to Newton’s law of gravitation.
• It uses Euler-Forward to discretize the motion.
• All planets are lined up at y=0 at t=0
• All planets are set to travel in the y-direction at t=0
Parameters

Object masses:

Mean distances from sun:

```matlab
% total number of big objects in solar system
Nsphere = 10;

% color of planets
Col = [1:Nsphere];

% Gravitational constant in kg^-2*m^3*s^-2
Grav = 6.67259*(10^(-11));

% mass of planets in kg
Mass = zeros(Nsphere,1);
Mass(1) = 0.33*(10^24);  % Mercury
Mass(2) = 4.87*(10^24);  % Venus
Mass(3) = 5.97*(10^24);  % Earth
Mass(4) = 0.642*(10^24); % Mars
Mass(5) = 1899*(10^24);  % Jupiter
Mass(6) = 568*(10^24);   % Saturn
Mass(7) = 86.8*(10^24); % Uranus
Mass(8) = 102*(10^24);  % Neptune
Mass(9) = 0.0125*(10^24); % Pluto
Mass(10) = 1047.3486*Mass(5); % The sun

% initial position in m
X   = zeros(Nsphere,1);
Y   = zeros(Nsphere,1);
X(1) = 57.9*(10^9);         % Mercury
X(2) = 108.2*(10^9);       % Venus
X(3) = 149.6*(10^9);       % Earth
X(4) = 227.9*(10^9);       % Mars
X(5) = 778.6*(10^9);       % Jupiter
X(6) = 1433.5*(10^9);      % Saturn
X(7) = 2872.5*(10^9);      % Uranus
X(8) = 4495.1*(10^9);      % Neptune
X(9) = 5870.0*(10^9);      % Pluto
X(10) = 0;                 % The sun

scatter(X,Y,'or');
```

Initial velocities of objects:

<table>
<thead>
<tr>
<th>Object</th>
<th>Initial Velocity (m/s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mercury</td>
<td>47.89*10^3</td>
</tr>
<tr>
<td>Venus</td>
<td>35.03*10^3</td>
</tr>
<tr>
<td>Earth</td>
<td>29.79*10^3</td>
</tr>
<tr>
<td>Mars</td>
<td>24.13*10^3</td>
</tr>
<tr>
<td>Jupiter</td>
<td>13.06*10^3</td>
</tr>
<tr>
<td>Saturn</td>
<td>9.64*10^3</td>
</tr>
<tr>
<td>Uranus</td>
<td>6.80*10^3</td>
</tr>
<tr>
<td>Neptune</td>
<td>5.40*10^3</td>
</tr>
<tr>
<td>Pluto</td>
<td>4.70*10^3</td>
</tr>
<tr>
<td>Sun</td>
<td>0</td>
</tr>
</tbody>
</table>
Set dt:

Time loop:

Calculate acceleration:

Advance X,Y,VX,VY

Plot the first four planets and the sun

end Time loop
Mercury has nearly completed its orbit. Data shows 88 days. Run for 3 more days and the simulation agrees!!!
Team Exercise

• Get the planets1.m file from the web site
• This scripts includes:
  – the mass of all planets and the sun
  – their mean distance from the sun
  – the mean velocity of the planets.
• Run the script, see how the planets run!
• Add a comet to the system (increase Nsphere etc.)
• Start the comet out near Jupiter with an initial velocity heading in system.
• Add a moon near the earth.
• Extra credit if you can make the comet loop the sun and hit a planet ☺
Next Lecture

• More accurate schemes

• More complicated ODEs

• Variable time step and embedded methods used to make sure errors are within a tolerance.