Today’s class

• Start looking at algorithms for the final project
• Two final project topics:
  – N-body numerical integration orbit problem
  – “Brick breaking game” which I call “brick-brack-brock”
• Both projects will be physics based but in the game project, user interaction and “game excitement” will need to be considered in the design.
• Today: Go through the physics first and then look at game interaction.
Physics in both projects

- Both projects will involve the effects of forces being applied to bodies.
- For the N-body project: Gravitational forces between orbiting bodies.
- For game project: We could be more adventurous and have strange forces that affect game play. At minimum the ball will be effected by a constant gravity force (a real trajectory for the ball) and the bricks should fall according to gravity. In this project, objects collide and so we have finite size bodies that need to be considered.
- Basic model will be $F=ma$ in both projects.

Class Project

- Runge-Kutta Method for numerical integration.
- There are many different forms of this numerical integration technique.
- The forms we need are those that solve the equation:
  - $y'' = f(x,y)$ where $x$ is an independent variable and $y$ is the dependent variable.
  - In our project, $x$ will be time and $y$ is position.
  - $y''$ is the second time derivative of position.
- There is also a form in which
  - $y'' = f(x,y,y')$
  - In this case, the acceleration depends on the velocity as well.
  - This could be useful if we want to include drag effects in the orbit integration.
- For program design we need to think about whether we want to add this or not.
- Decision will be based on, in practice, how difficult it is.
Runge-Kutta integration

• Compare the two versions of the second-order system
• $y'' = f(x,y,y')$

$$y_{n+1} = y_n + h[y_n' + \frac{1}{6}(k_1 + k_2 + k_3)] + O(h^5)$$

$$y_{n+1}' = y_n' + \frac{1}{6}[k_1 + 2k_2 + 2k_3 + k_4]$$

$k_1 = hf(x_n, y_n, y_n')$

$k_2 = hf(x_n + h/2, y_n + hy_n'/2 + hk_1/8, y_n' + k_1/2)$

$k_3 = hf(x_n + h/2, y_n + hy_n'/2 + hk_1/8, y_n' + k_2/2)$

$k_4 = hf(x_n + h, y_n + hy_n'/2 + hky_n'/2, y_n' + k_3)$

Form with no Velocity dependence

• $y'' = f(x,y)$

$$y_{n+1} = y_n + h[y_n' + \frac{1}{6}(k_1 + 2k_2)] + O(h^4)$$

$$y_{n+1}' = y_n' + \frac{1}{6}k_1 + \frac{2}{3}k_2 + \frac{1}{6}k_3$$

$k_1 = hf(x_n, y_n)$

$k_2 = hf(x_n + h/2, y_n + hy_n'/2 + hk_1/8)$

$k_3 = hf(x_n + h, y_n + hy_n'/2 + hky_n'/2)$
Integration continued

- Notice the way the problem is formulated.
- $h$ is the step size (i.e., the time step the integrator is going to use)
- In the first form of Runge-Kutta, the error in the $y_{n+1}$ is $O(h^5)$
- This means the error is “of the order of” $h$ to the 5th power.
- If we halve the step size $h$, then the error should be approximately 32 times smaller.
- (See the num_int.f routine where a similar relationship was used but in that case we could quantify the error)
- In the second form of Runge-Kutta, the error was $O(h^4)$ meaning that halving the step size would reduce the error by approximately 16 times.
- How are we going to use this? (Think about what was done in num_int.f).

Integration error

- In some cases, the actual size of the error in the numerical integration can be determined based on the high order derivative of the function (as was done in num_int.f and is done in the Matlab quad function)
- In many cases, these formulas are very complex and can take a long time to compute (relative to the numerical integration itself).
- You can obtain an approximate estimate of the error in the integration by looking at the difference between the results using two step sizes.
- The easiest implementation is to use $h$ and $h/2$ as the two step sizes.
- Using $h$, the integrator is step once to time $t+h$ and using $h/2$ two steps are taken to $t+h$
- If the difference in $y$ between $h$ and $h/2$ is $\Delta y$, then the error in the integrator with step size $h$ is $\Delta y = \Delta y/(1-1/2^n)$ where $n$ is 4 or 5 depending on the integrator used.
Integrator error continued

- Using this difference between the results with two step sizes, we can estimate the error in one step of the integrator.
- For long integrations, it is important to note the error is not random but tends to retain the same sign with each step and so will grow as the system is integrated.
- If the errors were random, their accumulated effects would grow as the square-root of the number of steps.
- A better approximation is the accumulated error will grow linearly with time.
- An estimate of the error is bounded between these values.
- Using this approach, we can monitor the error.
- (Sometimes this approach is used to correct the integrator, thus increasing its performance by another power of h --- but the error in corrected integrator is difficult is assess.
- What next in our design of the solution?

Other aspects to consider

- If we will possibly have a changing step size to keep the integration errors low, then what do we do about output of the results (or passing the results to an internal graphics routine)?
- The “result output” rate (i.e., the time interval for output the results) will differ from the integration step size most likely. How will be specify this?
- Options:
  - Have the user specify time interval for output
  - Have the user specify a distance that must travel before the results are output?
- Which is better? Pros and cons to each approach. If time given than maybe too many points are generated that can never really be displayed. If distance, then time step in not the same and an animation of the results will give the wrong impression of the speed the objects are moving with.
Basic “flow” of your program

• What blocks to be need in the program?
  – Input block
    • User must be able to give information about the position, velocity and mass of each body
    • User should give desired accuracy
    • User should give something about the output (could be file name and time step)
    • All these entries need checks to make sure that the values are legal.
  – Numerical integration block:
    • We need to implement the equations given earlier (which form to use?)
    • We need to be able to evaluate f(x,y,y') where x is time, y is position and y' is velocity

Program structure

– Numerical integration block (cont)
  • We need to check that the integration is meeting the tolerances given by the users.
– Output or graphics block
  • Depending on program used, will need either tabular output that can be used in an external graphics program or
  • We will generate the graphics directly in the program.
  • In either case, your program will generate tabular data so that the results can be checked.

• What else is needed?
  – A way to check the program. How are we going to do this?
**Program checking?**

- How do we check that this program works:
  - In this specific case: Use 1 massive body and 2 small bodies (so that they do not perturb each other)
  - Orbits in this case will be elliptical and we can use an analytic theory to verify orbit (Keplerian orbit)
  - Use approximate 3-body solutions (e.g., Earth-Moon-Sun system) to check characteristics
  - E.g. if the “moon” is put in an inclined orbit relative to the “Earth’s” orbit about the sun, the lunar orbit plane should precess. The rate of precession can be computed approximately analytically.
  - Compare results with another implementation (for example the Mac Planets program) — need to be very careful when doing this. Getting the same results does not mean you are correct and getting different results does not mean you are wrong.

**Brick-Brack-Brock design**

- Basic idea in this project is to have users hit a ball to knock bricks out of the ceiling. In most versions of this type of game, the bricks simply disappear but this violates conservation of mass. In our version the bricks will fall.
- Some ideas on the behavior of the game:
  - Bricks may require a certain amount of energy be imparted to them before they fall (the faster the ball is moving when it hits a brick the more energy it imparts
  - The bricks might attract the ball as it falls
  - The paddle may be “soft” so that energy is lost on the hits (vertical motion of the paddle could be used to add energy)
  - User input? Keyboard or arrow keys is common but no vertical motion. How about mouse input for 2-D motion with a ceiling so that you can not just sit close to the bricks?
More on design

- How should game start? Normally injection of ball at random direction and speed (with in limits). Since we will have vertical motion, the ball could start on the paddle and it could be “served” by the user?
- How does the game get more difficult with more levels? User’s paddle gets smaller? Ball gets lighter and faster? Bricks become more difficult to dislodge?
- Until we have a basic working model of the program, these decisions can be difficult to make so design the program to allow all possibilities.

Game timing and speed

- How to control the game speed? In the N-body integration, the integration step size can be reduced to maintain accuracy but in a game this could lead to non-uniform speed
- Question here is: Will the game need to be artificially slowed to allow human interaction (i.e., pauses added to slow it down -- duration of pauses could be reduced in higher levels to make the game go faster. Some experimentation may be needed here.
Basic game look

- Example below of how the game may look:

![Diagram of a game showing a moving ball, falling brick, user paddle, and color denoting energy level. Balls can be lost through top and bottom.]

Remainder class:

- For both projects: Starting writing down ideas of how each of the programs should operate
  - What are the main algorithms that will be needed
  - What should be the basic representation of different quantities (e.g., treating bricks, balls, paddles and planets as objects)
  - What will be the data and methods associated with each object.
  - Other issues that need to be addressed?