```
C = cross(A, B)
```

`cross` only works for 3-dimensional vectors; the result is a 3-dimensional vector.

A common use of `cross` is to compute torques. If you represent a moment arm R and a force F as 3-dimensional vectors, then the torque is just

```
Tau = cross(R, F)
```

If the components of `R` are in meters and the components of `F` are in Newtons, then the torques in `Tau` are in Newton-meters.

13.3 Celestial mechanics

Modeling celestial mechanics is a good opportunity to compute with spatial vectors. Imagine a star with mass m_1 at a point in space described by the vector P_1, and a planet with mass m_2 at point P_2. The magnitude of the gravitational force[‡] between them is

$$f_g = G\frac{m_1 m_2}{r^2}$$

where r is the distance between them and G is the universal gravitational constant, which is about $6.67 \times 10^{-11} Nm^2/kg^2$. Remember that this is the appropriate value of G only if the masses are in kilograms, distances in meters, and forces in Newtons.

The direction of the force on the star at P_1 is in the direction toward P_2. We can compute relative direction by subtracting vectors; if we compute `R = P2 - P1`, then the direction of `R` is from `P1` to `P2`.

The distance between the planet and star is the length of R:

```
r = norm(R)
```

The direction of the force on the star is $\hat{R}$:

```
rhat = R / r
```

Exercise 13.1 *Write a sequence of MATLAB statements that computes* `F12`*, a vector that represents the force on the star due to the planet, and* `F21`*, the force on the planet due to the star.*

Exercise 13.2 *Encapsulate these statements in a function named* `gravity_force_func` *that takes* `P1`*,* `m1`*,* `P2`*, and* `m2` *as input variables and returns* `F12`*.*

Exercise 13.3 *Write a simulation of the orbit of Jupiter around the Sun. The mass of the Sun is about* 2.0×10^{30} *kg. You can get the mass of Jupiter, its distance from the Sun and orbital velocity from* ***http://en.wikipedia.org/wiki/Jupiter****. Confirm that it takes about 4332 days for Jupiter to orbit the Sun.*

[‡]See `http://en.wikipedia.org/wiki/Gravity`

13.4 Animation

Animation is a useful tool for checking the results of a physical model. If something is wrong, animation can make it obvious. There are two ways to do animation in MATLAB. One is to use `getframe` to capture a series of images and `movie` to play them back. The more informal way is to draw a series of plots. Here is an example I wrote for Exercise 13.3:

```
function animate_func(T,M)
    % animate the positions of the planets, assuming that the
    % columns of M are x1, y1, x2, y2.
    X1 = M(:,1);
    Y1 = M(:,2);
    X2 = M(:,3);
    Y2 = M(:,4);

    minmax = [min([X1;X2]), max([X1;X2]), min([Y1;Y2]), max([Y1;Y2])];

    for i=1:length(T)
        clf;
        axis(minmax);
        hold on;
        draw_func(X1(i), Y1(i), X2(i), Y2(i));
        drawnow;
    end
end
```

The input variables are the output from `ode45`, a vector `T` and a matrix `M`. The columns of `M` are the positions and velocities of the Sun and Jupiter, so `X1` and `Y1` get the coordinates of the Sun; `X2` and `Y2` get the coordinates of Jupiter.

`minmax` is a vector of four elements which is used inside the loop to set the axes of the figure. This is necessary because otherwise MATLAB scales the figure each time through the loop, so the axes keep changing, which makes the animation hard to watch.

Each time through the loop, `animate_func` uses `clf` to clear the figure and `axis` to reset the axes. `hold on` makes it possible to put more than one plot onto the same axes (otherwise MATLAB clears the figure each time you call `plot`).

Each time through the loop, we have to call `drawnow` so that MATLAB actually displays each plot. Otherwise it waits until you finish drawing all the figures and *then* updates the display.

`draw_func` is the function that actually makes the plot:

```
function draw_func(x1, y1, x2, y2)
    plot(x1, y1, 'r.', 'MarkerSize', 50);
    plot(x2, y2, 'b.', 'MarkerSize', 20);
end
```

The input variables are the position of the Sun and Jupiter. `draw_func` uses `plot` to draw the Sun as a large red marker and Jupiter as a smaller blue one.

Exercise 13.4 *To make sure you understand how* `animate_func` *works, try commenting out some of the lines to see what happens.*

One limitation of this kind of animation is that the speed of the animation depends on how fast your computer can generate the plots. Since the results from `ode45` are usually not equally spaced in time, your animation might slow down where `ode45` takes small time steps and speed up where the time step is larger.

There are two ways to fix this problem:

1. When you call `ode45` you can give it a vector of points in time where it should generate estimates. Here is an example:

   ```
   end_time = 1000;
   step = end_time/200;
   [T, M] = ode45(@rate_func, [0:step:end_time], W);
   ```

   The second argument is a range vector that goes from 0 to 1000 with a step size determined by `step`. Since `step` is `end_time/200`, there will be about 200 rows in `T` and `M` (201 to be precise).

   This option does not affect the accuracy of the results; `ode45` still uses variable time steps to generate the estimates, but then it interpolates them before returning the results.

2. You can use `pause` to play the animation in real time. After drawing each frame and calling `drawnow`, you can compute the time until the next frame and use `pause` to wait:

   ```
   dt = T(i+1) - T(i);
   pause(dt);
   ```

   A limitation of this method is that it ignores the time required to draw the figure, so it tends to run slow, especially if the figure is complex or the time step is small.

Exercise 13.5 *Use* `animate_func` *and* `draw_func` *to vizualize your simulation of Jupiter. Modify it so it shows one day of simulated time in 0.001 seconds of real time—one revolution should take about 4.3 seconds.*

13.5 Conservation of Energy

A useful way to check the accuracy of an ODE solver is to see whether it conserves energy. For planetary motion, it turns out that `ode45` does not.

The kinetic energy of a moving body is $mv^2/2$; the kinetic energy of a solar system is the total kinetic energy of the planets and sun. The potential energy of a sun with mass m_1 and a planet with mass m_2 and a distance r between them is

$$U = -G\frac{m_1 m_2}{r}$$

Exercise 13.6 *Write a function called* `energy_func` *that takes the output of your Jupiter simulation,* `T` *and* `M`*, and computes the total energy (kinetic and potential) of the system for each estimated position and velocity. Plot the result as a function of time and confirm that it decreases over the course of the simulation. Your function should also compute the relative change in energy, the difference between the energy at the beginning and end, as a percentage of the starting energy.*

You can reduce the rate of energy loss by decreasing `ode45`'s tolerance option using `odeset` (see Section 12.1):

```
options = odeset('RelTol', 1e-5);
[T, M] = ode45(@rate_func, [0:step:end_time], W, options);
```

The name of the option is `RelTol` for "relative tolerance." The default value is `1e-3` or 0.001. Smaller values make `ode45` less "tolerant," so it does more work to make the errors smaller.

Exercise 13.7 *Run* `ode45` *with a range of values for* `RelTol` *and confirm that as the tolerance gets smaller, the rate of energy loss decreases.*

Exercise 13.8 *Run your simulation with one of the other ODE solvers MATLAB provides and see if any of them conserve energy.*

13.6 What is a model for?

In Section 8.2 I defined a "model" as a simplified description of a physical system, and said that a good model lends itself to analysis and simulation, and makes predictions that are good enough for the intended purpose.

Since then, we have seen a number of examples; now we can say more about what models are for. The goals of a model tend to fall into three categories.

prediction: Some models make predictions about physical systems. As a simple example, the duck model in Exercise 7.2 predicts the level a duck floats at. At the other end of the spectrum, global climate models try to predict the weather tens or hundreds of years in the future.

design: Models are useful for engineering design, especially for testing the feasibility of a design and for optimization. For example, in Exercise 12.6 you were asked to design the golf swing with the perfect combination of launch angle, velocity and spin.

explanation: Models can answer scientific questions. For example, the Lotka-Volterra model in Section 10.4 offers a possible explanation of the dynamics of animal populations systems in terms of interactions between predator and prey species.

The exercises at the end of this chapter include one model of each type.

13.7 Glossary

spatial vector: A value that represents a multidimensional physical quantity like position, velocity, acceleration or force.

norm: The magnitude of a vector. Sometimes called "length," but not to be confused with the number of elements in a MATLAB vector.

unit vector: A vector with norm 1, used to indicate direction.

dot product: A scalar product of two vectors, proportional to the norms of the vectors and the cosine of the angle between them.

cross product: A vector product of two vectors with norm proportional to the norms of the vectors and the sine of the angle between them, and direction perpendicular to both.

projection: The component of one vector that is in the direction of the other (might be used to mean "scalar projection" or "vector projection").

13.8 Exercises

Exercise 13.9 *If you put two identical bowls of water into a freezer, one at room temperature and one boiling, which one freezes first?*

Hint: you might want to do some research on the Mpemba effect.

Exercise 13.10 *You have been asked to design a new skateboard ramp; unlike a typical skateboard ramp, this one is free to pivot about a support point. Skateboarders approach the ramp on a flat surface and then coast up the ramp; they are not allowed to put their feet down while on the ramp. If they go fast enough, the ramp will rotate and they will gracefully ride down the rotating ramp. Technical and artistic display will be assessed by the usual panel of talented judges.*

Your job is to design a ramp that will allow a rider to accomplish this feat, and to create a physical model of the system, a simulation that computes the behavior of a rider on the ramp, and an animation of the result.

Exercise 13.11 *A binary star system contains two stars orbiting each other and sometimes planets that orbit one or both stars*[§]*. In a binary system, some orbits are "stable" in the sense that a planet can stay in orbit without crashing into one of the stars or flying off into space.*

Simulation is a useful tool for investigating the nature of these orbits, as in Holman, M.J. and P.A. Wiegert, 1999, "Long-Term Stability of Planets in Binary Systems," Astronomical Journal *117, available from* `http://citeseer.ist.psu.edu/358720.html`.

Read this paper and then modify your planetary simulation to replicate or extend the results.

[§]See `http://en.wikipedia.org/wiki/Binary_star`.