SCIENTIST 1:

1a) To find a stable separation between two atoms, we must find a distance between the two atoms in which the net force acting between the two atoms is equal to 0. We can then plug this point back into the original potential energy function given to determine the minimum value of the interatomic potential energy.

The negative slope of the potential energy graph as a function of time is equal to the net force, so if we find the derivative of the function, we will be able to find the point(s) that the slope is equal to 0.

$$U(r) = 4\epsilon\left\{\left(\frac{\sigma}{r}\right)^{12} - \left(\frac{\sigma}{r}\right)^{6}\right\}$$

$$F(r) = \frac{dU}{dr} = 4\epsilon\left\{-12\frac{\sigma^{12}}{r^{13}} + 6\frac{\sigma^{6}}{r^{7}}\right\}$$

$$F(r) = -4e\left\{\frac{12\sigma^{12}}{r^{13}} - \frac{6\sigma^{6}}{r^{7}}\right\}$$

Now, to find the possible point(s) where the net force is equal to 0, simply set F(r) to 0 and isolate r.

$$0 = -4\epsilon\left\{\frac{12\sigma^{12}}{r^{13}} - \frac{6\sigma^{6}}{r^{7}}\right\}$$

$$\frac{48\epsilon\sigma^{12}}{r^{13}} = \frac{24\epsilon\sigma^{6}}{r^{7}}$$

$$2\sigma^{6} = r^{6}$$

$$r = \pm\sqrt[6]{2\sigma^{6}} = \sqrt[6]{2}\sigma = \frac{-3.7}{4}\text{Å}$$

The reason the ± symbol was dropped was because you can't have a "negative" distance between two atoms, and since σ is known and positive we know the root term must also be positive. Now, plug this distance back into the potential energy function to find the minimum value of the interatomic potential energy.

$$U(r) = 4\epsilon\left\{\left(\frac{\sigma}{r}\right)^{12} - \left(\frac{\sigma}{r}\right)^{6}\right\}$$

$$U\left(\sqrt[6]{2}\sigma\right) = 4(18.2meV)\left\{\left(\frac{1}{\sqrt[6]{2}}\right)^{12} - \left(\frac{1}{\sqrt[6]{2}}\right)^{6}\right\}$$

$$U\left(\sqrt[6]{2}\sigma\right) = 4(18.2meV)\left(-\frac{1}{4}\right) = -18.2meV$$

Therefore, the most stable separation distance between the two atoms is $r = \frac{-3.7}{4}$Å, and the minimum value of the interatomic potential energy is $U = 18.2meV$.

1b) There are multiple advantages to using a computer to find solutions to this problem. For starters, computers make graphing a function like this really easy to do, so you can visually see how the interatomic potential energy value changes as the separation distance changes.

The biggest advantage to using a computer to computationally solve this problem is that because a computer can do thousands or even millions of calculations in a very short period of time, we can essentially "brute force" our way to finding the solution we are looking for. As seen above, it took nearly a whole page of calculations using calculus concepts (which could be difficult for someone not well versed mathematically to understand) to solve this problem.

Instead of solving this problem analytically, with a computer you can create a massive range of possible separation distance values and then you can make the computer plug every single one of these separation distance values into the function to determine which one outputs the smallest value. This method would be completely impractical for a human to do by hand, but since a computer can do so many calculations so quickly, it is often much quicker at producing a good approximation of the answer compared to the analytical method. The numerical method may not be quite as accurate as the more mathematically rigorous analytical method, but it gives a very good approximation and is much easier for somebody to conceptually understand the process that is actually occuring compared to the analytical method.

Another advantage of using a computer to solve a problem like this is that once code has been written, it is often very easy to translate to solve similar problems. For example, say you created a program that can solve the problem presented earlier, and then you wanted to investigate how the potential changed when you used two different kinds of atoms. If you solved this problem using the analytical method, you would have to completely restart the whole process and redo all those derivatives. But if you solve the problem computationally, it is as easy as swapping out any changed constants for the new atoms and then plugging in the new interatomic potential energy function, and then your program will immediately spit out the answers.

The advantages of solving this problem and problems like this computationally instead of analytically definitely outweigh the cons, especially if you are planning on studying intermolecular interactions for a long period of time for various types of atoms. The flexibility of a well designed program will increase your efficiency in your research because you will be able to quickly shift your program to observe any type of atom (assuming you know the parameters of their interaction), and the ability to quickly plot results from these programs makes using the computational method for problems like this an easy choice to make.

1c) The Bisection method and the Newton-Raphson method are both solid methods that will produce accurate results. However, I would definitely recommend using the Newton-Raphson method if at all possible.

The bisection method is much easier to understand mathematically, however in practice it often can take many many iterations to find a root, so it is very inefficient. The basic premise behind the bisection method is choosing a point that the function returns as a positive number and then choosing another point that makes the function return a negative number. Next, find the average (i.e. halfway point) between these two points. If this halfway point is positive, then set it as the first point and then find the new average between this new point and your other point. Similarly, if the halfway point is negative, set it as your second point and find the new average between this point and your other point. Repeating this process will continuously update those two points so the distance between them will get smaller and smaller so that slowly but surely the location of the root point is narrowed down.

The Newton-Raphson method on the other hand uses calculus to quickly find the roots of the function. The general "algorithm" or "formula" for the Newton-Raphson method is as follows:

(1) $$x = x_o - \frac{f(x_o)}{f'(x_o)}$$ (for finding a "root")

(2) $$x = x_o - \frac{f'(x_o)}{f''(x_o)}$$ (for finding a local minimum")

Formula (1) is used for finding a "root" of a function like a quadratic, while Formula (2) would be used for finding a local max/minimum of a function. Both of these methods work similarly. An initial approximation, x_o, is selected. Then whatever is returned from the function using this approximation is divided by whatever is returned from the derivative function at this point, and this quotient is subtracted from the initial approximation. If this process is repeated, x will approach the "root" rapidly. Formula (2) uses the exact same mathematical logic that Formula (1) uses, except it finds roots of the derivative function instead of the normal function like in Formula (1) . If you were to use the Newton-Raphson Method to solve this problem, you would need use Formula (2) to find the local (and global) minimum of the potential energy function, and therefore would need to find $U'(r)$ and $U''(r)$. After those are determined however, the code is extremely easy to put into place, and will produce a very accurate answer in an efficient manner.

While it is more mathematically rigorous to understand compared to the bisection method, the Newton-Raphson method is really easy to code once all of the necessary derivatives are calculated, and it is vastly more efficient compared to the bisection method because it requires far less iterations to get to the same solution. As a result, I would strongly recommend implementing the Newton-Raphson method into your program instead of the bisection method.

SCIENTIST 2:

2a) Of the three methods, I would definitely recommend the Runge-Kutta method. All three methods will produce pretty good results, however the Runge-Kutta method will produce much more accurate results. All three algorithms have a similar base idea: choose a given value to iterate time by, also known as Δt. Then calculate the position at $t + \Delta t$, and repeat this process. In general, all three algorithms become more and more accurate when Δt becomes smaller. However, where the three algorithms differ is how they actually calculate the position at $t + \Delta t$, and therefore the sensitivity for each algorithm for the Δt varies.

Euler's method is the most simple, and as a result is the most inaccurate. Euler's method is simply finding the acceleration (and therefore velocity and position) at $t + \Delta t$. This method does produce a decent approximation, however when the movement is complex such as in a scenario like tracking the position of an object in a driven harmonic oscillator like you are doing, the Δt value is simply too large compared to an infinitesimal value, and so much detail in the motion is lost. For Euler's method, the global truncation error is to the order of $O(\Delta t)$, which means as Δt is reduced, the error present in the final answer is also reduced by Δt.

The Trapezoid method is very similar to Euler's method. How this approximation works is that two points are chosen on a function, and then the area of a trapezoid that fits under a region connecting these two points. The smaller the distance between the two points, the higher the accuracy of the trapezoid method. This is because the trapezoidal method uses a linear approximation between the two points. These linear approximations better represent the curve over smaller distances, and lose accuracy over larger distances. The Trapezoid method is more accurate than Euler's method, with its global truncation error being the order of $O(\Delta t^2)$, meaning when Δt is halved, the error present in the final answer is reduced by $4*\Delta t$.

The Runge-Kutta method is more accurate then both of these methods mentioned above by a large margin. In this discussion, I will only discuss the fourth order Runge-Kutta method, as it is a more accurate and standard approximation than the second order Runge-Kutta method. How this method works is a starting point is imputed into the algorithm, and then the weighted average of four different values over an interval are added to the starting point to get the next point. The first value is given by the linear approximation of the function at the starting point. The second and third values are then given by taking a midpoint approximation of the point before it, and then the final point is found using the approximation of the previous value over the entire interval. The smaller the interval that is chosen originally, the more accurate the average of the slopes will be. This average is weighted because the second and third values are more accurate; their approximations are used over half the interval that both the first and fourth points are used. The global truncation error for the Runge-Kutta method is the order of $O(\Delta t^5)$, meaning when Δt is halved, the error present in the final answer is reduced by $32*\Delta t$. As a result, the Runge-Kutta method is pretty obviously the best method for you to use in your code.

2b) There is a large difference in accuracy between the second and fourth order Runge-Kutta algorithms. As discussed above, the fourth order Runge-Kutta algorithm has a global truncation error to the order of $O(\Delta t^5)$, whereas the second order Runge-Kutta algorithm is virtually the exact same as the trapezoidal method mentioned earlier, except the trapezoid is shifted and carried out at the midpoint of the chosen Δt making it slightly more accurate.

The main difference between the two methods is that the second order Runge-Kutta method finds the

2c) First, set all of your needed parameters, such as m, b, k, and F_0 as well as a Δt value. An initial position x also needs to be chosen. Next, you will need to set up a loop for the Runge-Kutta algorithm. I recommend a for loop with a large range. The loop needs to have this algorithm inside it:

$$k_1 = dt * f(x_0, y_0)$$
$$k_2 = dt * f(x_0 + \frac{1}{2}dt, y_0 + \frac{1}{2}k_1)$$
$$k_3 = dt * f(x_0 + \frac{1}{2}dt, y_0 + \frac{1}{2}k_2)$$
$$k_4 = dt * f(x_0 + dt, y_0 + k_3)$$
$$y_1 = y_0 + (\frac{1}{6})(k_1 + 2k_2 + 2k_3 + k_4)$$