Scientific Programming

Derivatives, Difference Equations and Euler's Method

1 Derivatives and Difference Equations

In calculus, the definition of the derivative is given by the limit

$$f'(x) = \lim_{h \to 0} \frac{f(x+h) - f(x)}{h}$$

Using Δx in place of h this becomes

$$f'(x) = \lim_{\Delta x \to 0} \frac{f(x+\Delta x) - f(x)}{h}$$

For small values of Δx, we get a good approximation

$$f'(x) \approx \frac{f(x+\Delta x) - f(x)}{h}$$

If we solve this equation for $f(x + \Delta x)$ we get the estimate

$$f(x+\Delta x) \approx f(x) + f'(x)\Delta x$$

Example If we know the derivative of a function, this can be used to approximate values of a function. For example, we can estimate the height, $h(t)$, of an object using its vertical velocity. Suppose, that at $t = 0$ an object has height of 10 meters and the objects vertical velocity is $v(t) = \sin(t^2)$.

Note: for other functions $v(t)$ its possible to use integral calculus to solve for $h(t)$ exactly. This is not straightforward for $v(t) = \sin(t^2)$.

We know that $h'(t) = v(t)$ and can use this to derive an update rule to estimate the height of the object.

$$\begin{aligned} h(t+\Delta t) &\approx h(t) + h'(t)\Delta t \\ &= h(t) + v(t)\Delta t \end{aligned}$$

We will let $\{t_i\}$ be the times that we are estimating the height at and $h_i \approx h(t_i)$ be the approximation of the height at time t_i. We will use the initial time 0 and height 10 to give initial values

$$\begin{aligned} t_1 &= 0 \\ h_1 &= 10 \end{aligned}$$

After each iteration $t_{i+1} = t_i + \Delta t$ and

$$\begin{aligned} h_{i+1} &\approx h(t_{i+1}) \\ &= h(t_i + \Delta t) \\ &\approx h(t_i) + v(t_i)\Delta t \\ &\approx h_i + v(t_i)\Delta t \end{aligned}$$

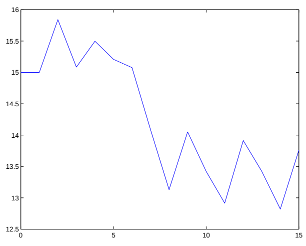
Combining everything we get the following iterative solution:

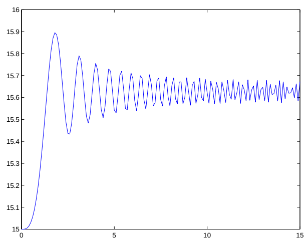
$$\begin{aligned} t_1 &= 0 \\ h_1 &= 10 \\ t_{i+1} &= t_i + \Delta t \\ h_{i+1} &= h_i + v(t_i)\Delta t \end{aligned}$$

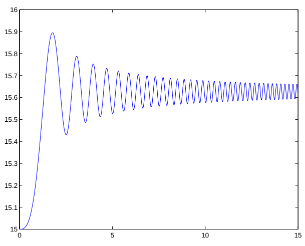
If we use $\Delta t = .1$ second and estimate the heights in the first 15 seconds using the velocity function $v(t) = \sin(t^2)$, we can use the following Matlab code:

```
dt = .1;
number_of_iterations = 15 / dt;

% Create arrays to store the times and heights
t = zeros(1,number_of_iterations+1)
h = zeros(1,number_of_iterations+1)

t(1) = 0;
h(1) = 10;
for i=1:number_of_iterations
  t(i+1) = t(i) + dt;
  h(i+1) = h(i) + sin(t(i)^2)*dt;
end
```

We can plot this solution using the command **plot**(t, h). More accurate solution can be found using smaller values of Δt. Below are the plots of height as a function of time with $dt = 1, .1$ and $.01$.







Notice that the graph gets smoother and more accurate with smaller time steps. Too many time steps take much more time and gets less additional accuracy.

2 Differential Equations: Population Model

The previous example can be thought of as a solution the the differential equation $\frac{dh}{dt} = \sin(t^2)$. If you are not familiar with a differential equation, it is an equation involving a function and its derivatives must satisfy.

Another example is the equation for population growth. Suppose $R(t)$ is the population of rabbits. If there an excess of food then the population of rabbits will grow and if there is not enough food the population will drop. The rate of change in the population can be modeled by the differential equation

$$\frac{dR}{dt} = kR(C - R)$$

where the constant k controls how quickly the population changes and the constant C, called the carrying capacity, is the maximum sustained rabbit population. If $0 < R < C$ then $\frac{dR}{dt} > 0$ so the rabbit population increases. And if $R > C$ then $\frac{dR}{dt} < 0$ and the rabbit population decreases.

Suppose that $R(t)$ is the population of rabbits (in thousands) in the year t and that $\frac{dR}{dt} = .1R(1 - \frac{R}{C})$. Suppose there are 3500 rabbits in January of 2000 and we want to estimate the population over the first 50 years of the 21st century. We get the update rules:

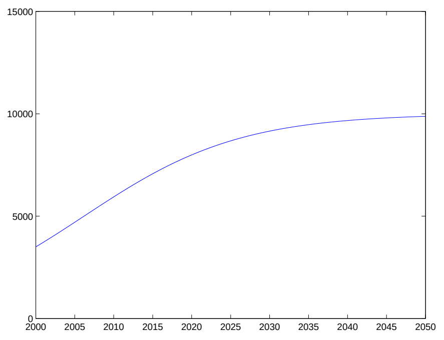
$$\begin{aligned} t_1 &= 2000 \\ R_1 &= 3500 \end{aligned}$$

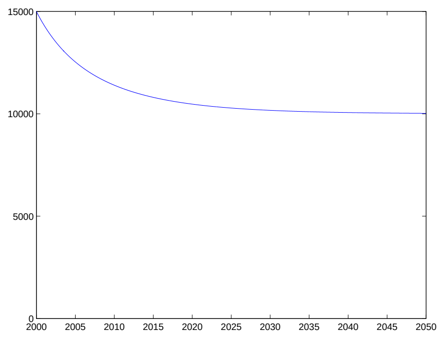
$$\begin{aligned} t_{i+1} &= t_i + \Delta t \\ R_{i+1} &= R_i + kR_i\left(1 - \frac{R_i}{C}\right) = R_i + .1R_i\left(1 - \frac{R_i}{10000}\right) \end{aligned}$$

In Matlab we get the following script:

```
% Do timesteps of 1 day
dt = 1/365
number_of_iterations = 50 / dt

% Create arrays to store the times and populations
t = zeros(1, number_of_iterations+1)
R = zeros(1, number_of_iterations+1)

t(1) = 2000
R(1) = 3500
for i=1:number_of_iterations
  t(i+1) = t(i) + dt
  R(i+1) = R(i) + .1 * R(i) * (1 - R(i)/10) * dt
end
```

Below are the graphs with initial rabbit population of 3,500 and 15,000.





Systems of Differential Equations: Predator-Prey A more realistic scenario for rabbits in the forest involve a predator, foxes. We can modify the differential equation for $\frac{dR}{dT}$ to take the fox population, F, into account. And add a new equation for how the fox population is affected by rabbits. Notice that if the fox population is zero then this simplifies to the previous equation.

$$\begin{aligned} \frac{dR}{dt} &= .04R\left(1 - \frac{R}{10000}\right) - .0005RF \\ \frac{dF}{dt} &= .00005RF - .2F \end{aligned}$$

The new terms involving the product RF incorporate the fact that the presence of both rabbits and foxes increases the fox population (since there is food available) and decreases the rabbit population. And the term $-.2F$ in the second equation account for natural death of the foxes.

There are now two dependent variables, R and F, to take into account. The update rules at each step become:

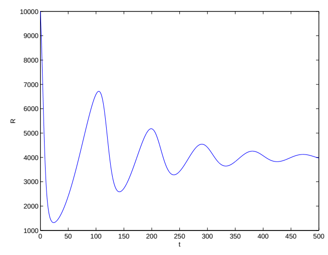
$$t_{i+1} = t_i + \Delta t$$

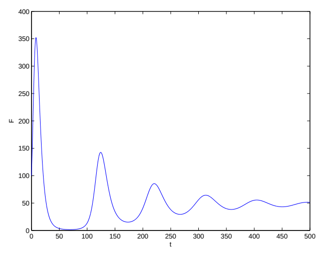
$$\begin{aligned}R_{i+1} &= R_i + .1R_i\left(1-\frac{R_i}{10000}\right)\Delta t\\ F_{i+1} &= F_i + (.00005R_iF_i - .2F_i)\Delta t\end{aligned}$$

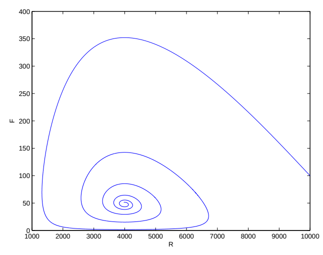
The Matlab code to simulate what happens when you begin with 10,000 rabbits and 100 foxes is below:

```
dt = .01;
n = 1000/dt;

t = zeros(1,n+1);
R = zeros(1,n+1);
F = zeros(1,n+1);

t(1) = 0;
R(1) = 10000;
F(1) = 100;

for i=1:n
  t(i+1) = t(i) + dt;
  R(i+1) = R(i) + (.04*R(i)*(1-R(i)/10000) - .0005*R(i)*F(i))*dt;
  F(i+1) = F(i) + (.00005*R(i)*F(i) - .2*F(i))*dt;
end
```

Below are three graphs: R as a function of t, F as a function of t and the trajectory of population of R and F.







Second Order Equation: Pulling a Bus When pulling an object friction pulls in the opposite direction. When moving at a velocity v the resistive force of friction is equal to kmv where k is the friction coefficient and m is the mass of the object. In strongmen competitions they pull large objects, like buses, down the street. Suppose the strongman pulls the bus with a constant force p then to total force is $F = p - kmv$. Using the fact that $F = ma$, we can solve for acceleration and get the differential equation

$$a = \frac{dx^2}{dt^2} = \frac{dx}{dt} = \frac{F}{m} = \frac{p-kmv}{m}$$

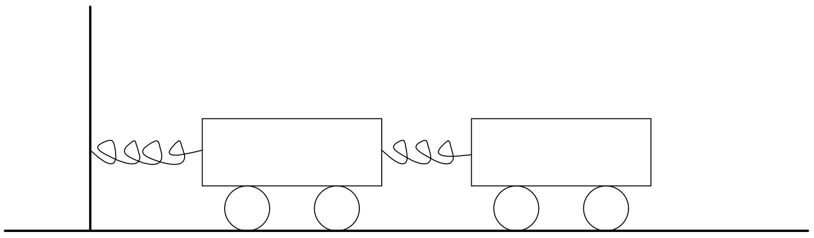
We can use this formula for acceleration to estimate velocity. The estimated velocity can then be used to estimate position. The update rules would be:

$$\begin{aligned}t_{i+1} &= t_i + \Delta t\\ v_{i+1} &= v_i + \frac{p-kmv_i}{m}\Delta t\\ x_{i+1} &= x_i + v_i\Delta t\end{aligned}$$

Suppose that the mass of the bus is 4,500 kg, the strongman pulls with a force of 500 N and the coefficient of friction is .04 N/(m/s) and that the bus is stationary. Graph the position of the bus as he pulls it for 30 seconds. The Matlab code to do this is below:

```
dt = .1;
n = 30/dt;

m = 4500;
k = .04;
p = 500;

t = zeros(1,n+1);
v = zeros(1,n+1);
x = zeros(1,n+1);

for i=1:n
  t(i+1) = t(i) + dt;
  v(i+1) = v(i) + (p - k*m*v(i))/m * dt;
  x(i+1) = x(i) + v(i) * dt;
end

plot(t,x)
```

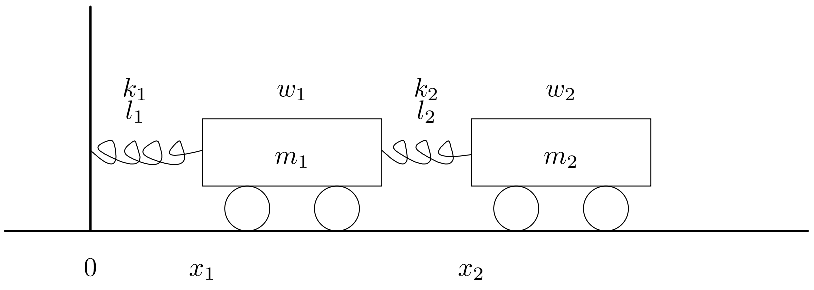
System of Second Order Equations: Double Springs We can use the same techniques to deal with multiple object. Suppose two cars are connected by springs to each other and the wall as below.



We will assume that there is not friction when the cars move and the wall is immovable. So the only forces involved will be from the spring. The force exerted by a spring is depends on three factors: its resting length (how long it would be if no forces we applied), how long it currently is and its spring constant. A spring with current length x, resting length l and spring constant k would push outward with a force of $k(l - x)$. So when the spring is pulled longer then its resting length is pulls back inward and when its shorter than its resting length is pushes outward.

Suppose x_1 and x_2 are the positions of the left end of the first car and second, respectively and the relevant constants are

Mass of 1st car	m_1	10 kg
Width of 1st car	w_1	.15 m
Resting length of 1st spring	l_1	.10 m
Spring constant of 1st spring	k_1	5 N/m
Mass of 2nd car	m_2	15 kg
Width of 2nd car	w_2	.15 m
Resting length of 2nd spring	l_2	.07 m
Spring constant of 2nd spring	k_2	2 N/m



The length of the first spring is x_1 so the outward force it exerts is equal to $F_1 = k_1(l_1 - x_1)$. The second spring's length is $x_2 - x_1 - w_1$, so exerts an outward force of $k_2(l_2 - x_2 + x_2 + w1)$.

Assume that initially both cars are motionless and pushed as far left as possible. (The first car is against the wall and the second pushed against the first.) We want to graph what happens for five minutes after the cars start moving.

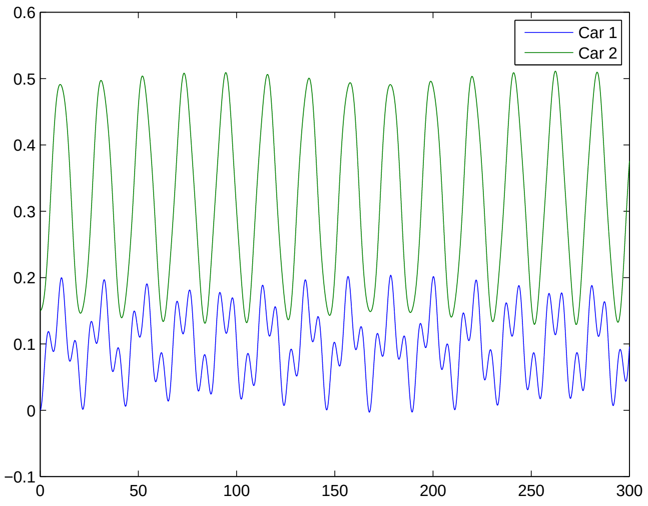
In our program, we will use the array $t, v1, x1, v2, x2$ to keep track of time, velocity of the 1st car, position of the 1st car, velocity of the 2nd car and position of the 2nd car, respectively. The Matlab script to do this is:

```
dt = .001;
n = 300/dt;

m1 = 10;
w1 = .15;
l1 = .10;
k1 = 5;

m2 = 15;
w2 = .15;
l2 = .07;
k2 = 2;

t = zeros(1,n+1);
v1 = zeros(1,n+1);
x1 = zeros(1,n+1);
v2 = zeros(1,n+1);
x2 = zeros(1,n+1);

t(1) = 0;
v1(1) = 0;
x1(1) = 0;
v2(1) = 0;
x2(1) = w1;

for i=1:n
  F1 = k1 * (l1 - x1(i));
  F2 = k2 * (l2 - x2(i) + x1(i) + w1);

  t(i+1) = t(i) + dt;

  v1(i+1) = v1(i) + (F1 - F2)/m1 * dt;
  x1(i+1) = x1(i) + v1(i) * dt;
```

```
    v2(i+1) = v2(i) + F2/m2 * dt;
    x2(i+1) = x2(i) + v2(i) * dt;
end

plot(t,x1, t,x2)
legend('Car 1', 'Car 2')
```

Notice in the code that the acceleration for the 1st car is `(F1 - F2)/m1`; this is because the first spring pushes the car to the right and the second pushes it to the left. The second car is only pushed right by the second spring so its acceleration is `F2/m2`. Also, note that dt is set to be .001 seconds. With such a rapidly moving object, such a small time step was necessary.

Below is the graph of the position:



Estimating Derivatives: Finding Velocity from Position Suppose we want to estimate the derivative of the function $f(x)$. The definition of the derivative says that $f'(x) = \lim_{h\to 0} \frac{f(x+h)-f(x)}{h}$. So estimate this we can plug in a small value for h. For example, if we have regularly spaced data in an array, use $h = \Delta x$. So $f'(x) \approx \frac{f(x+\Delta x)-f(x)}{\Delta x}$. This is called a forward difference. Basically, is approximates the derivative as the average rate of change in the interval after x.

If we have an array x that represents the positions of an object taken every Δt seconds, the we can use this to approximate the derivative. This derivative is the velocity of the object and we can create an array with this data using the formula v(i) = (x(i+1) − x(i)) / dt. Below is a loop implementation (assuming that the array x has n entries:

```
v = zeros(1,n);
for i=1:n-1
    v(i) = ( x(i+1) - x(i) ) / dt;
end
```

Note that the loop stops at n−1. If the loop continued to n then is would be asked to find x(n+1) which is past the end of the array.

An alternate way to calculate v is using array operations

```
v = ( x(2:n) - x(1:n-1) ) / dt;
```

In either case notice that v has one less entries than x. This is because there is no time interval after the last data point to use to approximate velocity.

It is also possible to approximate the derivative using the interval immediately prior to the point, called a backwards difference. This plugs in $h = -\Delta x$ into the derivative formula and gives the estimate:

$$f'(x) \approx \frac{f(x-\Delta x) - f(x)}{-\Delta x} = \frac{f(x) - f(x-\Delta x)}{\Delta x}$$

In Matlab this can be used to estimate the derivative using a loop

```
v = zeros(1,n);
for i=2:n
  v(i) = ( x(i) - x(i-1) ) / dt;
end
```

or array operations

```
v(2:n) = ( x(2:n) - x(1:n-1) ) / dt;
```

Notice that both fail to approximate the velocity at index 1, the beginning.

Both approximations miss a point. And the choice of using the interval immediately before or after seems arbitrary. An improved version that has less error is using the central difference:

$$f'(x) \approx \frac{f(x+\Delta x) - f(x-\Delta x)}{2\Delta x}$$

which is just the average of the forward and backward differences. This estimate has less error, but that is hard to show without doing a lot of calculus.

This formula cannot be used to estimate the derivatives at the beginning or end of the time interval, but we can use forward and backward differences, respectively, to do these estimates. Using a loop we get the following Matlab routine:

```
v = zeros(1,n);
v(1) = ( x(2) - x(1) ) / dt;
for i=2:n-1
  v(i) = ( x(i+1) - x(i-1) ) / (2*dt);
end
v(n) = ( x(n) - x(n-1) / dt;
```

Or using array operations:

```
v = zeros(1,n);
v(1) = ( x(2) - x(1) ) / dt;
v(2:n-1) = ( x(3:n) - x(1:n-2) ) / dt;
v(n) = ( x(n) - x(n-1) / dt;
```

Second derivatives: Acceleration from Velocity We can calculate the second derivative just by applying the above procedure twice. For example, to calculate acceleration we could do the following:

```
v = zeros(1,n);
v(1) = ( x(2) - x(1) ) / dt;
v(2:n-1) = ( x(3:n) - x(1:n-2) ) / dt;
v(n) = ( x(n) - x(n-1) / dt;

a = zeros(1,n);
a(1) = ( v(2) - v(1) ) / dt;
a(2:n-1) = ( v(3:n) - v(1:n-2) ) / dt;
a(n) = ( v(n) - v(n-1) / dt;
```

Or we could try to do this directly using the central difference formula:

$$\begin{aligned} f''(x) &\approx \frac{f'(x+\Delta x) - f'(x-\Delta x)}{2\Delta x} \\ &\approx \frac{\frac{f(x+2\Delta x)-f(x)}{2\Delta x} - \frac{f(x)-f(x-2\Delta x)}{2\Delta x}}{2\Delta x} \\ &= \frac{f(x+2\Delta x) - 2f(x) + f(x-2\Delta x}{4\Delta x^2} \end{aligned}$$

This formula only uses alternate values and can be improved to

$$f''(x) \approx \frac{f(x+\Delta x) - 2f(x) + f(x-\Delta x)}{\Delta x^2}$$

. This is the central difference approximation to the second derivative. Notice that this will not work at either end, but we can use backward and forward differences to get these approximations. These approximations are:

$$\begin{aligned} f''(x) &\approx \frac{f(x+2\Delta x) - 2f(x+\Delta x) + f(x)}{\Delta x^2} \\ f''(x) &\approx \frac{f(x) - 2f(x-\Delta x) + f(x-2\Delta x)}{\Delta x^2} \end{aligned}$$

The Matlab code to calculate the acceleration from position is:

```
a = zeros(1,n);
a(1) = ( x(3) - 2*x(2) + x(1)) / dt^2;
a(2:n-1) = ( x(3:n-1) - 2*x(2:n-1) + x(1:n-2) ) / (dt^2);
a(n) = ( x(n) - 2*x(n-1) + x(n-2) ) / dt^2;
```