

Examples

Below are some examples illustrating how to use difEQ

Example 1:

Say for some reason we need to solve the equation for $0 \leq t \leq 10$

$$\frac{d^2}{dt^2} \vec{r} + \begin{pmatrix} -.4 \\ .8 \\ .01 \end{pmatrix} \cdot \frac{d}{dt} \vec{r} + \begin{pmatrix} .3 \\ -.5 \\ -.1 \end{pmatrix} \cdot \vec{r} = \begin{pmatrix} \cos t \\ \cos t \\ \cos t \end{pmatrix}$$

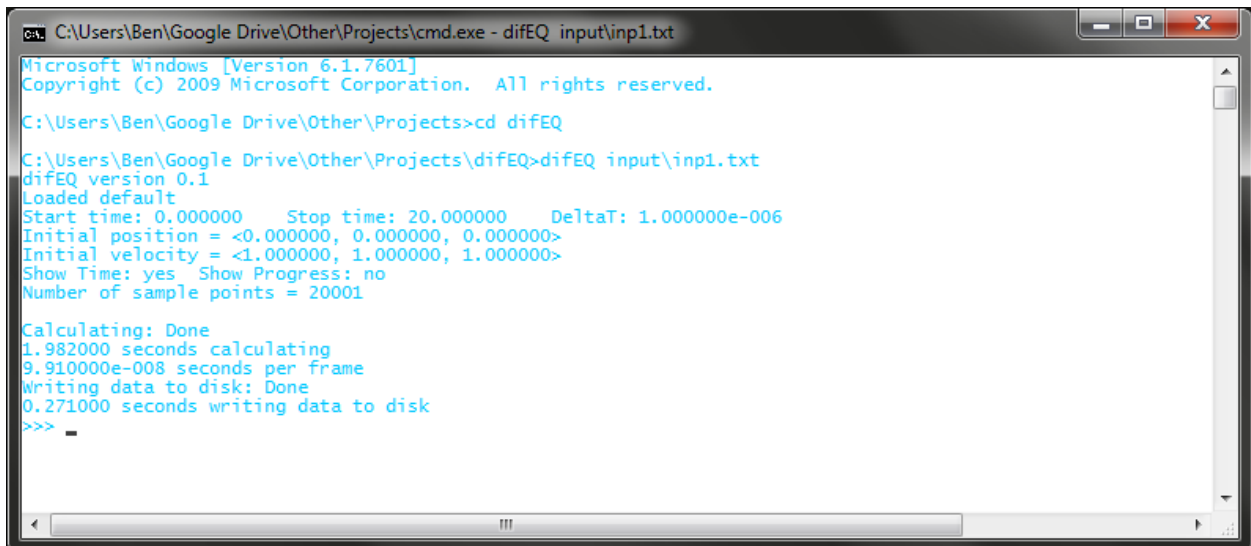
For the following initial conditions at $t = 0$:

$$\vec{r} = < 0, 0, 0 >$$
$$\frac{d}{dt} \vec{r} = < 1, 1, 1 >$$

It just so happens that this equation is already included in difEQ under the name “default”, so there is no need to write a new equation package, all we have to do is supply the proper input file. We’ll choose a time step of 10^{-8} and a store fraction of 10^6 . This means that we will simulate one billion frames and that we’ll store 100 of them per second of simulation time. Therefore the input file will be:

```
0, 20, 1E-6, 1E3
<0, 0, 0>
<1, 1, 1>
default
```

The “default” on the last line is actually optional since if no equation is specified then difEQ will automatically solve the default equation. If we save the file in the same directory as difEQ under “input\inp1.txt” then we simply need to execute the command “difEQ input\inp1.txt”, after printing some text and waiting a few seconds a python prompt should pop up.



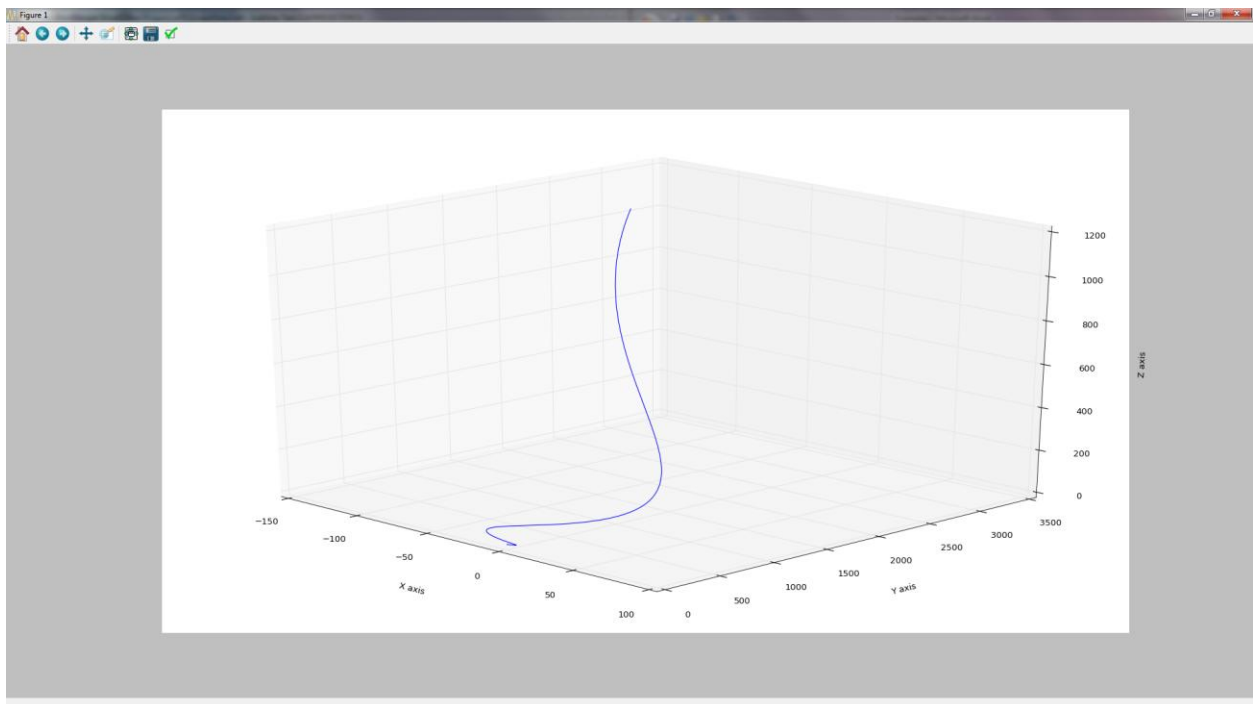
```
C:\Users\Ben\Google Drive\Other\Projects\cmd.exe - difEQ input\inp1.txt
Microsoft Windows [Version 6.1.7601]
Copyright (c) 2009 Microsoft Corporation. All rights reserved.

C:\Users\Ben\Google Drive\Other\Projects>cd difEQ
C:\Users\Ben\Google Drive\Other\Projects\difEQ>difEQ input\inp1.txt
difEQ version 0.1
Loaded default
Start time: 0.000000 Stop time: 20.000000 DeltaT: 1.000000e-006
Initial position = <0.000000, 0.000000, 0.000000>
Initial velocity = <1.000000, 1.000000, 1.000000>
Show Time: yes Show Progress: no
Number of sample points = 20001

Calculating: Done
1.982000 seconds calculating
9.910000e-008 seconds per frame
Writing data to disk: Done
0.271000 seconds writing data to disk
>>> -
```

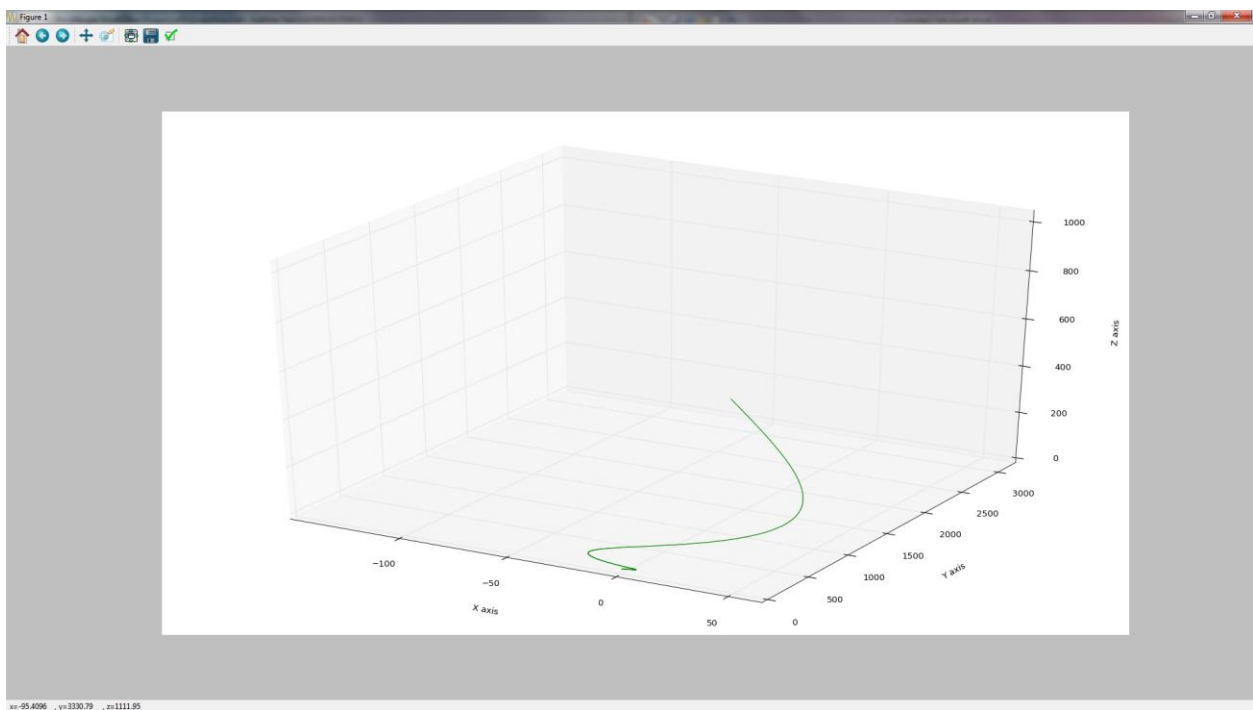
My terminal after executing “difEQ input\inp1.txt”

Great! Now we can finally graph the function. Simply enter `showGraph()` into the terminal and a 3D interactive graph will show up! You can drag the graph around with the left mouse button and zoom with the right.



The graph

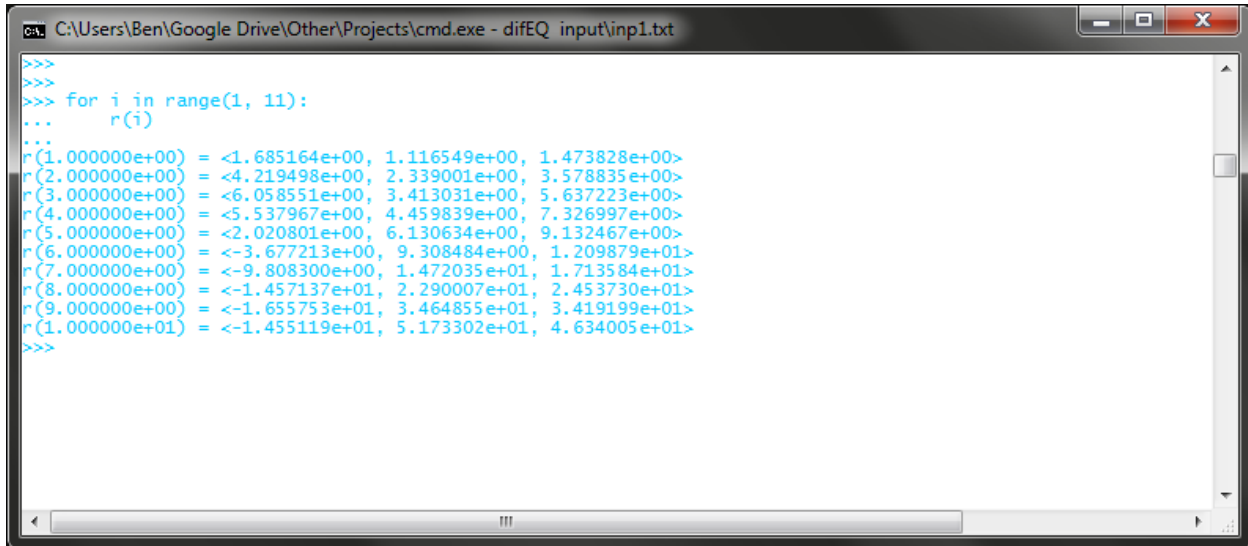
Say we want to see the position as a function of time, easy! Simply enter *animate(50)* (or simply *animate()* or *animate(someNum)*) and an animation will pop up.



In the middle of animation

Now say we want to find the position at $t = 1, 2, 3, \dots, 10$. While it's entirely feasible to simply enter $r()$ ten times, we can also write a simple script to accomplish the same thing. In the python terminal simply enter:

```
for i in range(1, 11):  
    r(i)
```



```
C:\Users\Ben\Google Drive\Other\Projects\cmd.exe - difEQ input\inp1.txt  
>>>  
>>>  
>>> for i in range(1, 11):  
...   r(i)  
...  
r(1.000000e+00) = <-1.685164e+00, 1.116549e+00, 1.473828e+00>  
r(2.000000e+00) = <-4.219498e+00, 2.339001e+00, 3.578835e+00>  
r(3.000000e+00) = <-6.058551e+00, 3.413031e+00, 5.637223e+00>  
r(4.000000e+00) = <-5.537967e+00, 4.459839e+00, 7.326997e+00>  
r(5.000000e+00) = <-2.020801e+00, 6.130634e+00, 9.132467e+00>  
r(6.000000e+00) = <-3.677213e+00, 9.308484e+00, 1.209879e+01>  
r(7.000000e+00) = <-9.808300e+00, 1.472035e+01, 1.713584e+01>  
r(8.000000e+00) = <-1.457137e+01, 2.290007e+01, 2.453730e+01>  
r(9.000000e+00) = <-1.655753e+01, 3.464855e+01, 3.419199e+01>  
r(1.000000e+01) = <-1.455119e+01, 5.173302e+01, 4.634005e+01>  
>>>
```

My terminal after querying.

Suppose that we now want to know all of the locations where $x=0$. We already know that this is true at $t=0$, but to find the other location we need to make use of conditions. To define a new condition that will find when $x=0$ simply enter the following into the terminal:

```
def zero_x(frame):  
    if frame < 0:  
        return float("NaN")  
    else:  
        return x_vals[frame]
```

Then it's a simple matter of calling `print_cnd(zero_x)` and the program will output all of the points where either $x=0$ or points near to where x passes through zero. The program will tell you which values are exact and which are approximate. If you wish to have more accurate answers simply decrease the store fraction so that the python program will have finer data to work with.

As is visible from my output below there are 4 points where the graph crosses the x axis. From looking at the graph this is obviously true. Apart from starting out on the x axis the graph proceeds to cross it three more times leading to a total of 4 points.

```
C:\Users\Ben\Google Drive\Other\Projects\cmd.exe - difEQ input\inp1.txt
>>>
>>>
>>> def zero_x(frame):
...     if frame<0:
...         return float("NaN")
...     else:
...         return x_vals[frame]
...
>>> print_cnd(zero_x)
exact match!
approx
approx
approx
r(0.000000e+00) = <0.000000e+00, 0.000000e+00, 0.000000e+00>
r(5.385000e+00) = <6.980000e-04, 7.129748e+00, 1.007674e+01>
r(1.159600e+01) = <2.099000e-03, 9.884039e+01, 7.443700e+01>
r(1.775200e+01) = <-2.437900e-02, 1.257831e+03, 5.090806e+02>
>>>
-
```

My output.