|  |  |  |  |
| --- | --- | --- | --- |
| **Objective** | To run examples from pySINDy | | |
| **Test** | **Result** | **Outputs** | **Comment/change** |
| Run jupyter notebook example 2 | Module not found error [all modules] | / | Try run as a .py file |
| Run example 2 in spyder | Module not found error (pysindy) | / | Add pysindy folder to path |
|  | Module not found error (cvxpy) | / | Download cvxpy – cloned from github |
|  | Module not found error (cvxpy) [same error] | / | Deleted and redownloaded – from conda command prompt |
|  | Ran example 2 successfully | (x)' = -2.000 x  (y)' = 1.000 y  (x)' = 0.772 sin(1 x) + 2.097 cos(1 x) + -2.298 sin(1 y) + -3.115 cos(1 y)  (y)' = 1.362 sin(1 y) + -0.222 cos(1 y) | Try another example |
| Run example 1 – basic usage section | Worked as shown in the notebook | (x0)' = -9.999 x0 + 9.999 x1  (x1)' = 27.992 x0 + -0.999 x1 + -1.000 x0 x2  (x2)' = -2.666 x2 + 1.000 x0 x1  Model score: 1.000000 | < predict derivatives w/ learned model  < Simulating forward in time |
| Run example 1 – different forms of input data section | / | / | Crashed a few times while I was editing the file (commenting out etc)  Look to see if another example is a bit shorter |
| Run a single section from example 3 – Lorenz system (where x and x dot has been measured and noise was added) | Ran successfully |  | Ran just as expected, will spend some time just looking at the code now |

|  |  |  |  |
| --- | --- | --- | --- |
| **Objective** | Try the lotka-volterra equations (SINDy in python) | | |
| **Test** | **Result** | **Outputs** | **Comment/change** |
| Copied from pysindy example 1 – change ‘lorenz’ to ‘lotka’ | error | / | Will change initial conditions [x0\_train] to have 2 values |
| Set initial conditions to 1500 and 100 | Output roughly correct equations – has extra constant terms | (x0)' = 0.024 1 + 49030455189.979 x0^2 + -735456827857.187 x0 x1  (x1)' = -159.237 1 + -48664764909.337 x0^2 + 729971473647.537 x0 x1 | Really big numbers ?  Try plotting to see if that tells us anything |
| Added plotting section of code | It did output a model, a model score and 2 plots – but they all look wrong | (x0)' = 0.024 1 + 49030455189.979 x0^2 + -735456827857.187 x0 x1  (x1)' = -159.237 1 + -48664764909.337 x0^2 + 729971473647.537 x0 x1  Model score: -15314586060973511016448.000000 | Will go through whats actually happening in the code  Change t\_train? |
| Instead of the time going from 0 – 10, make it 0 – 100 (scale more similar to the matlab version) | This made the output worse (changing the time) | / | Look at the lotka function?  so, what’s ‘p’? do I input this?  Try find example that uses it? |
| Write the equations rather than calling the function | Followed the very basic example (from the readme), got an error 🡪 | ValueError: Length of t should match x.shape[-2]. | Change t ? what else could I change |
| Changed initial conditions (from 1500 & 100, to 15 & 2) | Wrong output | (x0)' = -92.170 x0 + 5.466 x0^2 + -4.461 x0 x1  (x1)' = 95.682 x0 + -1.998 x1 + -5.614 x0^2 + 4.313 x0 x1  Model score: -44.492385 | Completely wrong equations and plot  3 terms? |
| *Following notes from meeting 29/06/2022* | | | |
| Rewrite function & small timestep   * Time now 1 (train and test) | Maybe looks closer? | (x0)' = 0.000  (x1)' = 0.361 x1  Model score: -6.827463 | Equations are too sparse but they’re the right magnitude – or at least closer |
| Slightly larger timestep Time now 2 (train and test) | Same as above | (x0)' = 0.000  (x1)' = 0.201 x1  Model score: -2.506456 | Same as above  Note: hadn’t actually imported new module so not sure how it worked |
| Same time step, now having imported the new module | Performed the same as previous | (x0)' = 0.000  (x1)' = 0.201 x1  Model score: -2.506456 |  |
| Larger time (50 [train & test]) | Numerical derivative looks closer, model still looks bad | (x0)' = 0.000  (x1)' = 0.000  Model score: -0.114569 |  |
| Large time – same as matlab, 300 [both again] | Model still just 0s, but seeing shape in the numerical derivative | (x0)' = 0.000  (x1)' = 0.000  Model score: -0.002624 | Why is x0 going into the negative? |

|  |  |  |  |
| --- | --- | --- | --- |
| **Objective** | Try the lotka-volterra equations (SINDy in python) – using SINDy-SA example | | |
| **Test** | **Result** | **Outputs** | **Comment/change** |
| Copied the parts from the predator prey sindy example fom the sindySA github that looked like the basic sindy process | It did successfully output equations but they were too sparse | (x)' = 0.000  (y)' = 0.188 x^2 y | Changed the sparsity parameter (0.1 🡪 0.001) |
|  | Equations not sparse enough | (x)' = -5257.577 x + 313.712 y + 2810.882 x^2 + 303.576 x y + 736.609 y^2 + -394.015 x^3 + 86.549 x^2 y + -684.689 x y^2 + 362.500 y^3  (y)' = -2361.088 y + 39.496 x^2 + 8597.427 x y + -32326.105 y^2 + -90.412 x^3 + -1014.013 x^2 y + 4046.646 x y^2 + 5492.544 y^3 | Try changing the lotka Volterra arguments to what I’ve been using |
|  | Equations too sparse (again) | (x)' = 0.000  (y)' = 0.000  UserWarning: Sparsity parameter is too big (0.01) and eliminated all coefficients | Sparsity parameter (threshold) 0.01 🡪 0.001 |
|  | Equations not sparse enough (again) |  | Sparsity parameter (threshold) 0.02 🡪 0.002,  Also changed format of lotkavolterra function [but didn’t make difference] |
|  | Incorrect equations | (x)' = 0.003 x^3  (y)' = -0.072 x y + 0.008 x^2 y + -0.005 x y^2 |  |

|  |  |  |  |
| --- | --- | --- | --- |
| **Objective** | Combining parts of all the examples | | |
| **Test** | **Result** | **Outputs** | **Comment/change** |
| Specifying the sindy model | Specifying lambda so it was less sparse resulted in closer equations, but x1 eqn is still too sparse | (x)' = 0.050 x + -0.005 x y  (y)' = -0.011 y  Model score: 0.505307    Ill conditioned matrix warning | Model fits x0 plot – but its still negative so it this the function I have to change?  Will change lambda again to try get x1 less sparse |
| Changed initial conditions to 15 & 100, and 10 & 300 | Similar results | (x)' = 0.050 x + -0.005 x y  (y)' = -0.022 y  Model score: 0.506125 | The x1 model actually looks like it has the shape a little bit? |
| Flipped the original initial conditions | The equations look closer, but the graph looks worse [?] | (x)' = 0.048 x + -0.005 x y  (y)' = 0.743 x + -0.200 y  Model score: 0.996942 |  |
| Increased run time |  | (x)' = 0.048 x + -0.005 x y  (y)' = 0.743 x + -0.200 y  Model score: 0.419014 |  |
| Set alpha and lamba (arguments in the optimizer fn) to 0 |  | (x)' = -0.281 x + 0.007 x^2 + -0.004 x y  (y)' = 0.034 x + -0.200 y + -0.001 x^2  Model score: 0.927593 |  |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Objective** | Restarting fresh – following pysindy example 3 2d linear ode example (rather than Lorenz example as before) | | | |
| **Test** | **Result** | **Outputs** | | **Comment/change** |
|  |  | **Reminder of goal** | (x0)' = 0.050 x0 + -0.005 x0 x1  (x1)' = -0.200 x1 + 0.005 x0 x1 |  |
| Running to the point of printing the model (no plots), set sparsity to 0 | Successfully out puts non-sparse equations, as expected | (x0)' = 0.002 x0 + -0.001 x1 + 0.009 x0^2 + -0.003 x0 x1 + 0.003 x1^2 + -0.001 x0 x1^2  (x1)' = 0.001 1 + 0.003 x0 + -0.003 x1 + 0.011 x0^2 + -0.008 x0 x1 + 0.019 x1^2 + -0.001 x0^3 + -0.003 x0^2 x1 + -0.004 x0 x1^2 | | Try incrementally increasing lambda (sparsity parameter) |
| Lambda = 0.0001 | Successfully output less sparse equations, but missing | (x0)' = 0.050 x0 + -0.005 x0 x1  (x1)' = -0.200 x1 | | Decrease lambda   * Lambda is smaller that it has to be for matlab |
| Lambda = 0.00001 | Really close, 1 wrong term | (x0)' = 0.050 x0 + -0.005 x0 x1  (x1)' = -0.200 x1 + 0.001 x0 x1 | | !! function was written wrong this has identified the correct system for the function (to 3 d.p.) !! Change b & c term to 0.005 [rather than 0.0005 as is in the notes – b/c it looks like the function rounds |
| Fix function (so it actually matches goal above) | Now is not sparse enough | (x0)' = -0.073 x0 + -0.005 x0 x1  (x1)' = -0.002 1 + 0.128 x0 + -0.200 x1 + 0.005 x0 x1 | | Note how lambda is proportional to smallest term in the equations  Increase lambda |
| Lambda = 0.0001 | Really close, but with an additional constant | (x0)' = 0.050 x0 + -0.005 x0 x1  (x1)' = -0.014 1 + -0.200 x1 + 0.005 x0 x1 | | Increase lambda again |
| Lambda = 0.00015 | Success <3 | (x0)' = 0.050 x0 + -0.005 x0 x1  (x1)' = -0.200 x1 + 0.005 x0 x1 | | Next step is to add plotting |
| Adding plotting bit of code | Not sure if this has plot what I think it should have? |  | | Increase time span |
| Run to 75 (3 x as long as previous) |  |  | |  |
| Flip initial conditions | Error – infinite error | (wrong equations and no graph) | | Scale down but keep that flipped ratio (i.e. so its went [1500, 100] 🡪[100, 1500] 🡪[1, 15] |
| Scale down initial conditions | Correct equations, |  | | Potentially looking right (the time plot at least), scale up time again |
| Run to 200 | Looks correct (and still correct equations) |  | | Looks correct – scale up time further, and phase plot looks correct too |
| Run to 500 | As expected <3 |  | | Next steps   * Add noise to data and then try recover dynamics * Edit library? * Try a sindy variant * - sindySA maybe? * Try with bearing equations |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Objective** | Try modelling the rolling bearing equations | | | |
| **Test** | **Result** | **Outputs** | | **Comment/change** |
| **function** |  |  |
| **w/ numbers** | (x0)' = 50 -66.7 x0  (x1)' = 233.3 -66.7 x1 | Note it’s a 1st order not 2nd b/c I don’t know how yet |
| Try coding equations as 1st order and with a constant for the sum terms | Very busy equations, but it did sure output something, and graphs | (x0)' = -1.437 1 + -1.162 x0 + -0.668 x1 + -0.937 x0^2 + -0.317 x0 x1 + -0.753 x0^3 + -0.078 x0^2 x1 + -0.603 x0^4 + 0.078 x0^3 x1 + -0.482 x0^5 + 0.176 x0^4 x1  (x1)' = -109.327 1 + -88.094 x0 + -26.157 x1 + -70.983 x0^2 + -20.987 x0 x1 + 0.578 x1^2 + -57.195 x0^3 + -16.839 x0^2 x1 + 0.464 x0 x1^2 + -0.006 x1^3 + -46.084 x0^4 + -13.510 x0^3 x1 + 0.372 x0^2 x1^2 + -0.005 x0 x1^3 + -37.131 x0^5 + -10.840 x0^4 x1 + 0.299 x0^3 x1^2 + -0.004 x0^2 x1^3 | | Jump up the sparsity parameter |
| Repeat but threshold changed from 0.00015 to 1 | Equation 1 is more sparse, but 2 still really busy | (x0)' = 47.574 1 + -63.436 x0  (x1)' = -108.756 1 + -37.843 x0 + 3758.084 x1 + -284.358 x0^2 + -19302.825 x0 x1 + 251.701 x0^3 + 36475.130 x0^2 x1 + -200.420 x0^4 + -30585.684 x0^3 x1 + -34.465 x0^5 + 9606.276 x0^4 x1 | | Keep increasing the threshold  The model looks accurate on the time plot, but not the phase plot |
| Repeat but threshold change from 1 to 2 | Equation 1 is zeroed out (too sparse), but eqn 2 is still busy | (x0)' = 0.000  (x1)' = -108.756 1 + -37.843 x0 + 3758.084 x1 + -284.358 x0^2 + -19302.825 x0 x1 + 251.701 x0^3 + 36475.130 x0^2 x1 + -200.420 x0^4 + -30585.684 x0^3 x1 + -34.465 x0^5 + 9606.276 x0^4 x1 | |  |
| Repeated for a range of thresholds between 1 and 2 | Threshold has to be ~ below 1.4, but equations kept looking the same as above | / | | Maybe issue is in the library – the library does contain the needed terms but maybe just has too many unnecessary terms |
| Repeat w/ threshold at 1, but library has smaller order polynomial terms (poly order = 2) |  | (x0)' = 101.225 1 + -190.661 x0 + 74.261 x0^2  (x1)' = -145.874 1 + -117.627 x0 + -41.306 x1 + -94.847 x0^2 + -33.135 x0 x1 + 1.430 x1^2 | | The model looks accurate (on the plots) again, but equations still not right |
| Repeat w/ increased lamba = 1.5 |  | (x0)' = 47.574 1 + -63.436 x0  (x1)' = -145.586 1 + -118.543 x0 + -128.503 x1 + -94.153 x0^2 + 75.500 x0 x1 | | Again, time plot looks accurate, phase plot does not |

Notes for looking at second order sindy

<https://www.pnas.org/doi/10.1073/pnas.1906995116>

going to have to build library that has derivatives (look at sindy-pi for this), and how to write the python function for this?

* Example of solving 2nd order ode in python <https://www.epythonguru.com/2020/07/second-order-differential-equation.html>

|  |  |  |  |
| --- | --- | --- | --- |
| **Objective** | To run SINDy-PI examples from pySINDy | | |
| **Test** | **Result** | **Outputs** | **Comment/change** |
| Run 2d example from example 9 | Replicated results | x0\_dot = -0.426 x0 + 0.008 x1 + -0.001 x1x1x1x1  x1\_dot = 0.239 x1 + -0.336 x1x1 + 0.090 x1x1x1x1 + -0.020 x1x1x1x1x1 + 0.001 x1x1x1x1x1x1 + 0.975 x0x1\_dot + 1.089 x1x1\_dot + -0.024 x1x1x1x1\_dot + -0.014 x1x1x1x1x1x1x1\_dot  [[all other values were 0s]] | This example has been replaced on the github with a higher order ODE example |
| Run solve higher-order ODE example from example 9 | Error with using PDELibrary | TypeError: \_\_init\_\_() got an unexpected keyword argument 'temporal\_grid' | Looking at the function PDELibrary, it indicates that we’re to use it if using SINDyPI with PDEs, this example has a system of ODEs (so set to None (or remove as this is default)) |
| Repeat above with temporal\_grid argument for PDELibrary set to None | Another error | TypeError: \_\_init\_\_() got an unexpected keyword argument 'implicit\_terms' | Flags if SINDy-PI is being used, so it seems like it should be true (as it is in the example) – maybe you have to have temporal grid set to something (not None) |
| Repeat above with implicit\_terms set to None | Another error | ValueError: Spatial grid and the derivative order must be defined at the same time | From function “The spatial grid for computing derivatives”  Look for where this is used in another example  spatial\_grid=x  where  x=X[128:-128] |
| Repeat above with spatial\_grid set to x\_train | Index error | names = "(" + feature\_names[i] + ")"  IndexError: list index out of range |  |

After meeting 04/07/2022: going to go back to just looking at solving the rolling bearing equations w/out using sindy, then go back to looking through pysindy to find the best method for applying them in SINDy-PI (probably SINDy-PI)

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Objective** | Solve the rolling bearing equations in python (w/out applying SINDy) | | | |
| **Test** | **Result** | | **Outputs** | **Comment/change** |
| Run example of 2nd order differential equation (link mentioned above) | Ran as expected | |  | Will now look at rolling bearing equations |
| Solve equations given in ‘raceway defects’ paper (already given as system of 1st orders) Diagram  Description automatically generated | Set e to 0, so this is probably correct [?] | |  | Set e nonzero (to try get more interesting dynamics) |
| Repeat above w/ e = 10 | It did output a graph, but also an error | | ODEintWarning: Excess work done on this call (perhaps wrong Dfun type). Run with full\_output = 1 to get quantitative information.  warnings.warn(warning\_msg, ODEintWarning) | Look at the paper again to try get values that aren’t in table given (the ones just guessed)  Print out odes aswell maybe ?  Or print solutions? |
| Repeat above, not changed anything | Different graph was printed, same error | | ODEintWarning: Excess work done on this call (perhaps wrong Dfun type). Run with full\_output = 1 to get quantitative information.  warnings.warn(warning\_msg, ODEintWarning) |  |
| Repeat a few times to see if graph keeps changing, try different run times | Getting a diff result most times, sometimes the top left reappears | |  | Look at paper again and try changing some variables in the equations |
| Changing Q values | For Qx, Qy = 0 |  | | Something weird happens when time set to 10 – look different from at any other time ? |
| For Qx, Qy = 100 |  | |
| For Qx, Qy = 1000 |  | |
| Solve using equations given in original rolling bearing paper |  | |  | Split up into 2 functions, one for the x and one for the y (had issues getting float not subscriptable error when tried as 1 function)  I’ve oversimplified the equations |
| Rewrote the equations including the sin and cos terms in the R equations   * Started w/ just the x function | Looks like the raceway defects paper attempts (see above) | |  | Some of the constants were taken from the paper (d, D1, mu, c, Fx, Fy, m) some I made up (D2, K, alpha)  Is the angle for cos rad or degrees?   * It is in radians   Maybe its too simple to not have the sum terms?   * Add in a t term so it changed w time maybe |
| Add in a t to the cos term so it changes w time | This didn’t look like it changed anything | | Same as above | Add in the sum terms to the equations   * Loop ? * Manually type |

From 05/07 meeting: looking at plotting the different terms in the rolling bearing equations and then adding them in to the rolling bearing equations, to then solve them

|  |  |  |  |
| --- | --- | --- | --- |
| **Objective** | Break down the rolling bearing equations, understanding the dynamics of each term | | |
| **Test** | **Result** | **Outputs** | **Comment/change** |
| Solving smaller sections of the rolling bearing equations – plot the damping force (looking at the original rolling bearing paper)   * 1st order ODE | Expected to have a constant -ve slope, performed as expected |  | Did what I thought it would |
| The sum Rx   * First w/ constant values for displacement | Returned a value for the sum of Rx as expected |  | Checked that the loop works okay for the summing  Do I have to put that structure with in the function  Not a dfferential equation tho ? |
| Build an expression for the equations   * Looking at nonlinear model & sim paper | Can print an expression for Fx | Fx = 8.52456371201663e-11\*(-x - 1.0\*y - 1.41421356237309e-5)\*\*1.5 | Next step to plot it |
|  |  |  |  |
|  |  |  |  |
| Solve the ODEs considering 1 ball, so no sum terms   * Considering original paper | Initially plotted wrong thing |  | Unsure why there’s 2 lines ?   * Forgot to just plot 1 column like its as if we’ve plotted x’ vs time and x vs time |
|  | This (if I done it correctly) is the solution of |  | Plotted over smaller timescale to see in more detail (nonlinear only at the start?) |
|  |  |  |  |

Starting again, putting together what I’ve done, note file names b/c its got a bit messy

Papers:

1. Simulation and Analysis of Vibration of Rolling Bearing
2. Nonlinear model and simulation of a rolling bearing
3. Effect of the raceway defects …

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Test** | **Paper/form of equation** | **Filename** | **Result** | **Outputs** | **Comment/change** |
| Linear representation – replacing the complex terms with constants | 1/ | Rolling bearing 1ball, sect 1 | a +ve slope |  | Behaved as expected  (Fot and R are constant terms) |
| Solving for x and y (w/ linear terms) in a single function | 1/ | 2nd order ODEs, sect 4 |  |  | In this version Fot and R and linear functions of x and y |
| Solving for x and y, not linear because cos & sin terms in expression – but still just substituting in constants (not expanding Q) | 3/  Diagram  Description automatically generated | 2nd order ODEs, sect 6 | Gives weird spike at start (note running for longer just continues as a horizontal line) |  | Had been using incorrect notation for squaring  Not sure about spike |
| Repeating above but specifying a (small) step size | | |  |  | Smaller step size lets you see the dynamics better, the oscillation |
| Sections of the equations – damping force only | 1/ | Rolling bearing broke down, sect 1 | A -ve slope |  | Set the force in the y axis > force in the x therefore steeper slope as expected |
| Sections of the equations – the nonlinear contact forces | 2/ | Rolling bearing broke down, sect 4 | Printed an equation for Fx | Fx = 8.52456371201663e-11\*(-x - 1.0\*y - 1.41421356237309e-5)\*\*1.5 | Set up a loop to iterate the delta j term and summed to get Fx |
| Simplified equation – having 1 term instead of a sum term (as if there was 1 ball) | 1/    Where | 2nd order ODEs, sect 2 (consider moving?) | Doesn’t seem correct but not actually sure |  | So I’ve in theory tried to so the same thing twice but I’ve got very different plots – so what’s different   * I’ve included a t in the cos term for this one * I’ve also used 21.94 and 30 for D1, D2 |
| Simplified equation – having 1 term instead of a sum term (as if there was 1 ball)  [same as above] | 1/ | Rolling bearing 1ball, sect 2 | Different from above? |  | Here I’ve used 21.94e-3 and 30e-3 for D1, D2 |
|  |  |  |  |  |  |
| repeating simplified equation – combining the above 2 tests | 1/ | 2nd order ODEs, sect 2 (consider moving?) | Still different |  | Changed the guessed parameters in the first version to be the same as those in the 2nd version, kept longer run time   * Ive left out the power of 1.5 for the delta |
| Now matching |  | Now they’re the same   * The rolling bearing 1 ball one is neater |

|  |  |  |
| --- | --- | --- |
| Just playing about with parameters on the simplified model – considering 1 ball (from paper 1) | | |
| **Change** | **Outcome** | **Comment** |
| D2: 30e-3 🡪 40e-3 |  | Looks the same (linear) as its scaled (eg running to 0.01, 0.1, 1, 100)   * Scale of the x values is much bigger tho |
| D2: back to 30e-3  Alpha: pi/8 🡪 pi/2 |  | Only shows nonlinearity at very short time |
| Alpha: black to pi/8  K: 0.1 🡪 10 |  |  |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Test** | **Paper/form of equation** | **Filename** | **Result** | **Outputs** | **Comment/change** |
| Build a loop to sum sigma Rn | 1/    Where | Rolling bearing broke down, sect 3 | Calculated Rn iterating increasing alpha (angle of | Rx 1 = 24.03264  Rx 2 = 60.014992345414115  Rx 3 = 102.47403921660697  Rx 4 = 144.95074476377545  Rx 5 = 180.98338476377546  Rx 6 = 205.09128555741964  Rx 7 = 213.6092069316582  Rx 8 = 205.24534171653056  Rx 9 = 181.27798171653058  Rx 10 = 145.36090937111646  Rx 11 = 102.96714249992361  final sum Rx = 102.96714249992361 | Substituted x and y for a constant to simplify |
| Adapt the loop to considering x and y | 1/    Where | Rolling bearing broke down, sect 3 |  | Rx 1 = 12.0\*x + 0.03264  Rx 2 = 23.0865543901354\*x + 4.59220118838108\*y + 0.06528  Rx 3 = 31.571835764374\*x + 13.0774825626196\*y + 0.09792  Rx 4 = 36.1640369527551\*x + 24.1640369527551\*y + 0.13056  Rx 5 = 36.1640369527551\*x + 36.1640369527551\*y + 0.1632  Rx 6 = 31.571835764374\*x + 47.2505913428905\*y + 0.19584  Rx 7 = 23.0865543901354\*x + 55.7358727171291\*y + 0.22848  Rx 8 = 12.0\*x + 60.3280739055102\*y + 0.26112  Rx 9 = 7.105427357601e-15\*x + 60.3280739055102\*y + 0.29376  Rx 10 = -11.0865543901354\*x + 55.7358727171291\*y + 0.3264  Rx 11 = -19.571835764374\*x + 47.2505913428905\*y + 0.35904  final sum Rx = -19.571835764374\*x + 47.2505913428905\*y + 0.35904 | Would need to consider a y and alpha as functions of time to be able to plot ?  Or change alpha to 2pi/11 (makes the balls evenly spaced ? |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |

|  |  |  |  |
| --- | --- | --- | --- |
| **Objective** | Check using lotkavolterra that the odeint is doing what I think it should | | |
| **Test** | **Result** | **Outputs** | **Comment/change** |
| Solved the lotkavolterra function with odeint | Plotting rabbits and foxes against time looked correct, |  | Im happy that I can follow the indexing used in the functions  Note I had to change the initial conditions and run time  Phase plot looks funny, maybe a number of time steps issue (as in not enough) |
| Repeated but defined number of steps for time | Fixed phase plot as I had assumed |  |  |