# pytrajectory Documentation

***Release 0.3.2***

**Andreas Kunze**

April 07, 2014

CONTENTS

CHAPTER

ONE

# PYTRAJECTORY USER'S GUIDE

## 1.1 About PyTrajectory

PyTrajectory is a Python library for the determination of the feed forward control to achieve a transition between desired states of a nonlinear control system.

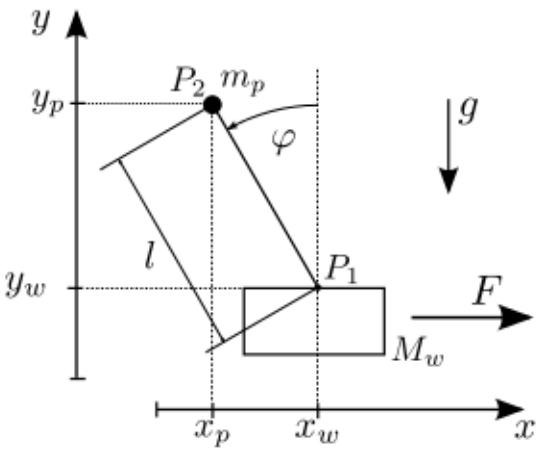
It's based an a study work written at the Technical University Dresden.

## 1.2 Usage

## 1.3 Examples

### 1.3.1 Translation of the inverse pendulum

An example often used in literature is the inverse pendulum. Here a force $F$ acts on a cart with mass $M_w$. In addition the cart is connected by a massless rod with a pendulum mass $m_p$. The mass of the pendulum is concentrated in $P_2$ and that of the cart in $P_1$. The state vector of the system can be specified using the carts position $x_w(t)$ and the pendulum deflection $\varphi(t)$ and their derivatives.




With the *Lagrangian Formalism* the model has the following state space representation where $u_1 = F$ and $x =$

$[x_1, x_2, x_3, x_4] = [x_w, \dot{x}_w, \varphi, \dot{\varphi}]$

$$
\begin{aligned}
\dot{x}_1 &= x_2 \\
\dot{x}_2 &= \frac{m_p \sin(x_3)(-l x_4^2 + g\cos x_3)}{M_w l + m_p \sin^2(x_3)} + \frac{\cos(x_3)}{M_w l + m_p l \sin^2(x_3)} u_1 \\
\dot{x}_3 &= x_4 \\
\dot{x}_4 &= \frac{\sin(x_3)(-m_p l x_4^2 \cos(x_3) + g(M_w + m_p))}{M_w l + m_p \sin^2(x_3)} + \frac{\cos(x_3)}{M_w l + m_p l \sin^2(x_3)} u_1
\end{aligned}
$$

A possibly wanted trajectory is the translation of the cart along the x-axis (i.e. by $0.5m$). In the beginning and end of the process the cart and pendulum should remain at rest and the pendulum should be aligned vertically upwards ($\varphi = 0$). As a further condition $u$ should start and end steadily in the rest position ($u(0) = u(T) = 0$). The operating time here is $T = 1[s]$.

```
# import trajectory class and necessary dependencies
from pytrajectory.trajectory import Trajectory
from sympy import sin, cos
import numpy as np

# define the function that returns the vectorfield
def f(x,u):
    x1, x2, x3, x4 = x        # system state variables
    u1, = u                   # input variable

    l = 0.5      # length of the pendulum rod
    g = 9.81     # gravitational acceleration
    M = 1.0      # mass of the cart
    m = 0.1      # mass of the pendulum

    s = sin(x3)
    c = cos(x3)

    ff = np.array([                       x2,
                   m*s*(-l*x4**2+g*c)/(M+m*s**2)+1/(M+m*s**2)*u1,
                                          x4,
              s*(-m*l*x4**2*c+g*(M+m))/(M*l+m*l*s**2)+c/(M*l+l*m*s**2)*u1
                  ])
    return ff

# boundary values at the start (a = 0.0 [s])
xa = [  0.0,
        0.0,
        0.0,
        0.0]

# boundary values at the end (b = 1.0 [s])
xb = [  0.5,
        0.0,
        0.0,
        0.0]

# create trajectory object
T = Trajectory(f, a=0.0, b=1.0, xa=xa, xb=xb)

# run iteration
T.startIteration()

# show results
```

```
T.plot()
```

CHAPTER

TWO

# PYTRAJECTORY MODULES REFERENCE

PyTrajectory is a Python library for the determination of the feed forward control to achieve a transition between desired states of a nonlinear control system.

## 2.1 `trajectory` Module

**class** `pytrajectory.trajectory.`**`Trajectory`**(*ff*, *a=0.0*, *b=1.0*, *xa=None*, *xb=None*, *g=None*, *sx=5*, *su=5*, *kx=2*, *delta=2*, *maxIt=7*, *eps=0.01*, *tol=1e-05*, *algo='leven'*, *use_chains=True*)

Base class of the PyTrajectory project.

**Parameters**

- **ff** (*callable*) – Vectorfield (rhs) of the control system
- **a** (*float*) – Left border
- **b** (*float*) – Right border
- **xa** (*list*) – Boundary values at the left border
- **xb** (*list*) – Boundary values at the right border
- **g** (*list*) – Boundary values of the input variables
- **sx** (*int*) – Initial number of spline parts for the system variables
- **su** (*int*) – Initial number of spline parts for the input variables
- **kx** (*int*) – Factor for raising the number of spline parts for the system variables
- **delta** (*int*) – Constant for calculation of collocation points
- **maxIt** (*int*) – Maximum number of iterations
- **eps** (*float*) – Tolerance for the solution of the initial value problem
- **tol** (*float*) – Tolerance for the solver of the equation system
- **algo** (*str*) – Solver to use
- **use_chains** (*bool*) – Whether or not to use integrator chains

**`DG`**(*c*)

This is the callable function that returns the jacobian matrix of the collocation system.

**`G`**(*c*)

This is the callable function that represents the collocation system.

**analyseSystem**()
Analyses the systems structure and sets values for some of the method parameters.

**buildEQS**()
Builds the collocation equation system.

**checkAccuracy**()
Checks whether desired accuracy for the boundary values was reached.

**dx**(*t*)
This function returns the left hand sites state at a given (time-) point `t`.

**getGuess**()
This method is used to determine a starting value (guess) for the solver of the collocation equation system –> docu p. 24

**initSplines**()
This method is used to initialise the temporary splines

**iterate**()
This method is used to run one iteration

**plot**()
Just calls `plot()` function from `tools`

**setCoeff**()
This method is used to create the actual splines by using the numerical solutions to set up the coefficients of the polynomial spline parts of every created spline.

**simulate**()
This method is used to solve the initial value problem.

**solve**()
This method is used to solve the collocation equation system.

**startIteration**()
This is the main loop –> [5.1]

**u**(*t*)
This function returns the inputs state at a given (time-) point `t`.

**x**(*t*)
This function returns the system state at a given (time-) point `t`.

## 2.2 `spline` Module

**class** `pytrajectory.spline.`**CubicSpline**(*a=0.0*, *b=1.0*, *n=10*, *tag=''*, *bc=None*, *bcd=None*, *bcdd=None*, *steady=True*)
This class provides an object that represents a cubic spline ...

**Parameters**

- **a** (*float*) – Left border of spline interval.
- **b** (*float*) – Right border of spline interval.
- **n** (*int*) – Number of polynomial parts the spline will be devided into.
- **tag** (*str*) – The 'name' of the spline object.
- **bc** (*tuple*) – Boundary values for the spline function itself.

- **bcd** (*tuple*) – Boundary values for the splines 1st derivative
- **bcdd** (*tuple*) – Boundary values for the splines 2nd derivative
- **steady** (*bool*) – Whether or not to call `makesteady()` when instanciated.

**dddf** (*x*)
This is just a wrapper for `evalf()` to evaluate the splines 3rd derivative.

**ddf** (*x*)
This is just a wrapper for `evalf()` to evaluate the splines 2nd derivative.

**df** (*x*)
This is just a wrapper for `evalf()` to evaluate the splines 1st derivative.

**evalf** (*x*, *d*)
bla bla bla

**Parameters**

- **x** (*float*) – The point to evaluate the spline at
- **d** (*int*) – The derivation order

**f** (*x*)
This is just a wrapper for `evalf()` to evaluate the spline itself.

**makesteady** ()
This method sets up and solves equations that satisfy boundary conditions and ensure steadiness and smoothness conditions of the spline in every joining point.

**set_coeffs** (*c_sol*)
This function is used to set up numerical values for the spline coefficients.

**Parameters** **c_sol** (*dict*) – Dictionary with coefficient and numerical value

**tmp_dddf** (*x*)
This is just a wrapper for `tmp_evalf()` with the splines 3rd derivative.

**tmp_ddf** (*x*)
This is just a wrapper for `tmp_evalf()` with the splines 2nd derivative.

**tmp_df** (*x*)
This is just a wrapper for `tmp_evalf()` with the splines 1st derivative.

**tmp_evalf** (*x*, *d*)
This function returns a matrix and vector to evaluate the spline or a derivative at x by multiplying the matrix with numerical values of the independent variables and adding the vector.

**Parameters**

- **x** (*real*) – The point to evaluate the spline at
- **d** (*int*) – The derivation order

**Returns** Matrix and vector that represent how the splines coefficients depend on the free parameters.

**Return type** tuple

**tmp_f** (*x*)
This is just a wrapper for `tmp_evalf()` with the spline itself.

pytrajectory.spline.**fdiff** (*func*)
This function is used to get the derivative of of a callable splinefunction.

## 2.3 solver Module

**class** pytrajectory.solver.**Solver**(*F*, *DF*, *x0*, *tol=0.01*, *maxx=10*, *algo='leven'*)

This class provides solver for the collocation equation system.

**Parameters**

- **F** (*callable*) – The callable function that represents the equation system
- **DF** (*callable*) – The function for the jacobian matrix of the eqs
- **x0** (*numpy.ndarray*) – The start value for the sover
- **tol** (*float*) – The (absolute) tolerance of the solver
- **maxx** (*int*) – The maximum number of iterations of the solver
- **algo** (*str*) – The solver to use

**gauss**()

**leven**()

This method is an implementation of the Levenberg-Marquardt-Method to approximatively solve a system of non-linear equations by minimizing

$\|F'(x_k)(x_{k+1} - x_k) + F(x_k)\|_2^2 + \mu^2\|x_{k+1} - x_k\|$

**newton**()

**solve**()

This is just a wrapper to call the chosen algorithm for solving the equation system

## 2.4 simulation Module

**class** pytrajectory.simulation.**Simulation**(*ff*, *T*, *start*, *u*, *dt=0.01*)

This class does something ...

**Parameters**

- **ff** (*callable*) – Vectorfield of the control system
- **T** (*float*) – Simulation time
- **u** (*callable*) – Function of the input variables
- **dt** (*float*) – Time step

**calcStep**()

**rhs**(*t*, *x*)

**simulate**()

## 2.5 utilities Module

**class** pytrajectory.utilities.**BetweenDict**(*d={}*)

Bases: dict

**class** pytrajectory.utilities.**Grid**

**class** pytrajectory.utilities.**IntegChain**(*lst*)
This class provides a representation of a integrator chain consisting of sympy symbols ...

**Parameters** **lst** (*lst*) – Ordered list of elements for the integrator chain

**predecessor**(*elem*)
This method returns the predecessor of the given element of the integrator chains, i.e. it returns $\int [elem]$

**Parameters** **elem** (*sympy.Symbol*) – An element of the integrator chain

**successor**(*elem*)
This method returns the successor of the given element of the integrator chains, i.e. it returns $\frac{d}{dt}[elem]$

**Parameters** **elem** (*sympy.Symbol*) – An element of the integrator chain

**class** pytrajectory.utilities.**Modell**

**draw**(*x*, *phi*, *frame*, *image=0*)

pytrajectory.utilities.**blockdiag**(*M*, *bshape=None*, *sparse=False*)
Takes block of shape `bshape` from matrix `M` and creates block diagonal matrix out of them.

**Parameters**

- **M** (*numpy.ndarray*) – Matrix to take blocks from
- **bshape** (*tuple*) – Shape of one block
- **sparse** (*bool*) – Whether or not to return created matrix as sparse matrix

### Examples

```
>>> A = np.ones((4, 2))
>>> print A
[[ 1.  1.]
 [ 1.  1.]
 [ 1.  1.]
 [ 1.  1.]]
>>> B = blockdiag(A, (2, 2))
>>> print B
[[ 1.  1.  0.  0.]
 [ 1.  1.  0.  0.]
 [ 0.  0.  1.  1.]
 [ 0.  0.  1.  1.]]
```

pytrajectory.utilities.**plot**(*sim*, *H*, *fname=None*)
This method provides graphics for each system variable, manipulated variable and error function and plots the solution of the simulation.

**Parameters**

- **sim** (*tuple*) – Contains collocation points, and simulation results of system and input variables
- **H** (*dict*) – Dictionary of the callable error functions
- **fname** (*str (optional)*) – If not None, plot will be saved as <fname>.png

**class** pytrajectory.utilities.**struct**

## 2.6 log Module

pytrajectory.log.**IPS**(*loc=None*)

**class** pytrajectory.log.**Logger**(*fname*, *mode*, *suppress*)

**write**(*text*)

**class** pytrajectory.log.**Timer**(*label='~'*, *verbose=True*)

pytrajectory.log.**err**(*text*, *lvl=0*)

pytrajectory.log.**info**(*text*, *lvl=0*)

pytrajectory.log.**logtime**(*text*, *lvl=0*)

pytrajectory.log.**msg**(*label*, *text*, *lvl=0*)

pytrajectory.log.**set_file**(*fname='/usr/bin/sphinx-build2_140407-200320.log'*, *suppress=False*)

pytrajectory.log.**warn**(*text*, *lvl=0*)

CHAPTER

THREE

# INDICES AND TABLES

- *genindex*
- *modindex*
- *search*

## p

## T



## U



## W



## X