A sage script entitled “NFT.PY” which computes generalized jet schemes

Andrew R. Stout[a,1,]

[a] *Department of Mathematics, Borough of Manhattan Community College, City University of New York, 199 Chambers Street, 10007, New York, NY, United States*

Abstract

The point of this note is to create an additional repository for a sage script which computes the equations defining generalized jet spaces.

1. Introduction

The SageMath code below (which was originally written for an older version of Sage see [2]) was used in his 2014 PhD thesis (cf. [7]) and in [6] and [5] to directly compute the equations of generalized jet schemes. This code is also available at the author’s github site: `https://github.com/infinity-groupoids/NFT.py` The author has recently checked that this code works SageMath version 10.6 using Python 3.12.5. This code is formally entitled NFT.py v1.0.0.0

Since this code calls many Singular (cf. [1]) procedures, the author has developed a new updated and faster singular library (cf. [3]) which is used in the paper [4]. This new Singular library is preferred by the author who has the intention to include this as a standard Singular library at some point. This new code is also available at the authors github site: `https://github.com/infinity-groupoids/arc.lib`

Updates of both programs will be documented on the author’s website `https://infinity-groupoids.github.io/index.html` and the author’s blog `https://infinity-groupoids.github.io/blog.html`

2. The code

Below we post the code for reference.

```
import sys
import datetime
import operator
from sage.all import *
from sage.symbolic.expression_conversions import PolynomialConverter

## ########################################################
#
# Andrew's stuff
#
## ########################################################


## ########################################################
## Class to organize methods and storing data variables
## ########################################################
class Space:
```

Email address: `astout@bmcc.cuny.edu` (Andrew R. Stout)

```
    def __init__(self):
        self.numvars = 0
        self.numeqs = 0
        self.firstequation = 0
        self.fatvars = 0
        self.fateqs = 0
        self.firstfatequation = 0
        return

    def setEquations(self):
        print("Creating functions for your space...")
        return

    def setFatEquations(self):
        print("Creating functions for your fat point...")
        return

    def toString(self):
        msg = "Symbols: " + str(self.numvars) + "\t"
        msg = msg + "Equations: " + str(self.numeqs) + "\n"
        return msg

    def toFatString(self):
        msg = "Symbols: " + str(self.fatvars) + "\t"
        msg = msg + "Equations: " + str(self.fateqs) + "\n"
        return msg


## ##########################################################
## Class to organize methods and storing data variables
## ##########################################################


## ##########################################################
## Helper methods
## ##########################################################
def getInt(msg):
    my_input = input(msg)
    try:
        return int(my_input)

    except:
        print("Input should be an integer, please try again")
        return getInt(msg)


## ##########################################################
def debug(msg):
    now = datetime.datetime.now()
    msg = "[" + str(now) + "] " + str(msg)
    print(msg)
    return


## ##########################################################
## Begin main program
## ########################################################
if __name__ == '__main__':

    mySpace = Space()
    mySpace.numvars = getInt("How many variables are in this space? ")
    mySpace.numeqs = getInt(
        "How many defining equations does your space have? ")

    print("Defining ambient space...")
    Poly1 = PolynomialRing(QQ, "x", mySpace.numvars)
    print(Poly1)
    Poly1.inject_variables()
    mySpace.setEquations()

    debug(mySpace.toString())

    print(
        'Using the variables above, input the expression for your first Equation
           and press return.'
    )
    mySpace.firstequation = SR(input())
    f = [];
    f.append(mySpace.firstequation)

    for i in range(1, mySpace.numeqs):
        print(
            'Using the variables above, input the expression for your next
               Equation and press return.'
        )
        mySpace.nextequation = SR(input())
        f.append(mySpace.nextequation)

    print('Check that the list of equations is correct:')
    print(f)

    mySpace.fatvars = getInt("How many variables are in this fat point? ")
    mySpace.fateqs = getInt(
        "How many defining equations does your fat point have? ")

    print("Defining ambient space...")
    Poly2 = PolynomialRing(QQ, "y", mySpace.fatvars)
    print(Poly2)
    Poly2.inject_variables()
    mySpace.setFatEquations()

    debug(mySpace.toString())

    print(
        'Using the variables above, input the expression for your first Equation
           of your Fat point and press return.'
    )
    mySpace.firstfatequation = SR(input())
    g = []
    g.append(mySpace.firstfatequation)

    for i in range(1, mySpace.fateqs):
```

```
    print(
        'Using the variables above, input the expression for your next
            Equation of your Fat point and press return.'
    )
    mySpace.nextfatequation = SR(input())
    g.append(mySpace.nextfatequation)
    I = ideal(g)

debug(mySpace.toFatString())

SingPoly2 = singular(Poly2)
singular.setring(SingPoly2.ring())
G = [str(g[i]) for i in range(mySpace.fateqs)]
J = singular.ideal(G)  #given by input
J = J.groebner()
B = list(J.kbase())

length = len(B)
C = [B[i].sage() for i in range(length)]

arcvars = length * mySpace.numvars
debug("Defining ambient space for your arc space...")

arcvars = length * mySpace.numvars
hh = mySpace.numvars + mySpace.fatvars + arcvars
Poly3 = PolynomialRing(QQ, "a", hh)
Poly3.inject_variables()
LL = list(Poly3.gens())
LL1 = [LL[i] for i in range(mySpace.numvars)]
LL2 = [
    LL[i]
    for i in range(mySpace.numvars, mySpace.numvars + mySpace.fatvars)
]
LL3 = [LL[i] for i in range(mySpace.numvars + mySpace.fatvars, hh)]
w = Poly2.gens()
Dict2 = {w[i]: LL2[i] for i in range(mySpace.fatvars)}
E = [C[i].subs(Dict2) for i in range(length)]
v = Poly1.gens()
Dict1 = {v[i]: LL[i] for i in range(mySpace.numvars)}
F = [f[i].subs(Dict1) for i in range(mySpace.numeqs)]

M = matrix(length, mySpace.numvars, LL3)
N = matrix(1, length, E)
D = N * M

DD = D.list()
Dict2 = {LL1[i]: DD[i] for i in range(mySpace.numvars)}
FF = [F[i].subs(Dict2) for i in range(mySpace.numeqs)]
idealF = ideal(FF)
debug(idealF)

tempJ = list(J)
lll = len(tempJ)
JJ = [tempJ[i].sage() for i in range(lll)]
w = Poly2.gens()
Dict2 = {w[i]: LL2[i] for i in range(mySpace.fatvars)}
```

```
tempI = [JJ[i].subs(Dict2) for i in range(lll)]
II = ideal(tempI)
debug(II)

QR = QuotientRing(Poly3, II)
QR.inject_variables()
pi = QR.cover()

p = [
    PolynomialConverter(FF[i], base_ring=QQ)
    for i in range(mySpace.numeqs)
]
rr = [p[i].symbol(FF[i]) for i in range(mySpace.numeqs)]
RR = [pi(rr[i]) for i in range(mySpace.numeqs)]
debug("going to factor ring")
d = [RR[i].lift() for i in range(mySpace.numeqs)]
debug("lifting to the cover")

debug("Computing tempL")
tempL = []
for i in range(mySpace.numeqs):
    j = 0
    for j in range(length - 1):
        cc = d[i].quo_rem(E[j])
        #debug("CC: " + str(cc))
        CC = list(cc)
        tempL = tempL + [CC[0]]
        a = simplify(d[i] - CC[0] * E[j])
        if (d[i] == a):
            debug("No change")
        #del d[i]
        #debug("d[i] prior to change: " + str(d[i]))
        d[i] = a
        #debug("d[i] after change: " + str(d[i]))
        #d.insert(i,a)
        j = j + 1
bigL = tempL + d

debug("... processing ...")
quoL = [pi(bigL[i]) for i in range(len(bigL))]
newL = [quoL[i].lift() for i in range(len(bigL))]
#runL=[factor(newL[i]) for i in range(len(newL))]

## What is tryL??
breadth = int(mySpace.numeqs)
depth = int(length)

tryL = []
## Initialize the list to -1
for i in range(breadth):
    j = 0
    for j in range(depth):
        tryL.append("NaN")

debug("... performing division ...")
## Populate list with real data
```

```
    for i in range(breadth):
        j = 0
        for j in range(depth):
            idx = (i * depth + j)
            tryL[idx] = list(newL[idx].quo_rem(E[j]))[0]

    #tryL=[list(newL[i].quo_rem(E[i]))[0] for i in range(len(newL))]
    #finL=[factor(tryL[i]) for i in range(len(bigL))]

    debug("Create ideal...")
    tempIdeal = Poly3.ideal(LL1 + LL2 + newL)
    debug(tempIdeal)
    debug(length)
    debug("Quotient ring")
    finQR = Poly3.quotient_ring(tempIdeal)
    finQR.inject_variables()

    debug(">> Equations for Arc space: ")

    debug("Singular")
    SingfinQR = singular(finQR)
    #SingredQR=singular(redQR)
    #nI=0*finQR

    debug(SingfinQR)
    #div=singular(nI)
    #nil=div.radical()
    #debug( ">> Equations for reduced Arc space: " )
    #debug( nil )
    #debug( SingredQR )

    ## Test code to remove redundant variables
    #ddd=Poly3.gens()
    #listd=list(ddd)
    #augddd=listd[len(LL1)+len(LL2):hh]
    #Poly4 = PolynomialRing(QQ, augddd)
    #Poly4.inject_variables()
```

References

[1] Decker, W.; Greuel, G.-M.; Pfister, G.; Schönemann, H.: SINGULAR 4-4-0 — A computer algebra system for polynomial computations. https://www.singular.uni-kl.de (2024). (cited on page 1)

[2] SageMath, *the Sage Mathematics Software (Version 6.2.Beta1)*, The Sage Developers, 2014, http://www.sagemath.org. (cited on page 1)

[3] Andrew Stout, *arc.lib: A Singular library for generalized jet schemes computations*, Version 1.0.0, June 2025. Available at arxiv.org (cited on page 1)

[4] Andrew Stout, *On a new singular library for computing generalized jet schemes with fast partial reduction and selected applications*, Pre-print on arxiv.org, 2025. (cited on page 1)

[5] Andrew Stout, *Jet schemes over auto-arc spaces and deformations of locally complete intersection varieties*, J. Algebra, vol 659, 2024, pp. 361–394. (cited on page 1)

[6] Andrew Stout, *On the auto igusa-zeta function of an algebraic curve*, J. Symb. Comput., vol. 79, 2017, pp. 156–185. (cited on page 1)

[7] Andrew Stout, *Motivic Integration over Nilpotent Structures*, Ph.D. Thesis, City University of New York, 2014, 179 pp. ISBN: 978-1321-30704-7, ProQuest LLC. (not cited)