arc.lib: A Singular library for generalized jet schemes computations

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 singular library which computes the equations defining generalized jet spaces and allows for partial reduction.

1. Introduction

In original work of the author's PhD Thesis [8] from 2014, the author produced SageMath (cf. [3]) code which computes the equations which define a generalized jet scheme. 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

Note that this code was additionally used in [6] and [7]. We also note that there has been recent work in Macaulay2 for computing classical jet spaces [2] for which it seems that the author's in that work are unaware of the previous work of the author of this note dating back to 2014. Therefore, it is advisable to make these items more widely known and available in case there is further work in this area. Perhaps, the authors of that work or even other authors will extend their Macaualay2 to the generalized jet scheme case.

Regardless, since the Sage script of the author (cf. [4]) calls many Singular (cf. [1]) procedures, the author has developed a new updated and significantly faster singular library (code included below) which is used in the paper [5]. 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

Note that the need for high order calculations with a standard computer is needed for dealing with many problems related to non-reduced jet space of fat points. It was impossible to investigate complex problems with the older Sage code due to memory issues. Now, many questions can actually be approached without the use of a supercomputer. Also, this new singular library has the ability to do a partial reduction which is enough – one does not necessarily need or even want full reduction to test theoretical work.

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.

```
////////////////////////////////////////////////////////////
version="arc.lib 1.0.0.0 Jun_2025 ";
category="Algebraic geometry";
// summary description of the library
info="
```

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

```
LIBRARY:   arc.lib  A Template for a Singular Library
AUTHOR:    Andrew Stout, email: astout@bmcc.cuny.edu

SEE ALSO:  standard_lib, Libraries,
           Typesetting of help and info strings

KEYWORDS: Jet schemes, arc.lib; Generalized jet schemes, arc.lib;

PROCEDURES:
  arc(ideal, ideal, ideal, list)        return list of equations of a generalized
      arc space relative to an embedded fat point

";
////////////////////////////////////////////////////////////////////////


// Main entry point with just 4 parameters
proc arc(ideal Vars, ideal I, ideal J, list optList)
{
    option(noredefine);

    // Determine whether to use the optional skip list
    int useOptList = 1;
    if (size(optList) == 0)
    {
        useOptList = 0;
    }
    else
    {
        for (int k = 1; k <= size(optList); k++)
        {
            if (optList[k] < 1)
            {
                useOptList = 0;
                break;
            }
        }
    }
    // Call arc_internal with explicit type conversion
    int useOptList_int = int(useOptList);
    arc_internal(Vars, I, J, optList, useOptList_int);
}

// Internal procedure without test functionality
proc arc_internal(ideal Vars, ideal I, ideal J, list optList, int useOptList)
{
    option(noredefine);

    // Compute and store global count at start of arc_internal
    int N = size(Vars);
    int M = size(optList);

    if (J == 0) { print("Error: The ideal J is the zero ideal. Terminating.");
        return; }

    // Step 2: identify fat point variables
```

```
ideal fatVars = (0);
int flag1 = 0;
poly currentVar;
int jdx1;
for (int idx1 = 1; idx1 <= N; idx1++)
{
flag1 = 0;
currentVar = var(idx1);

for (jdx1 = 1; jdx1 <= size(J); jdx1++)
{
        intvec w = 0:N;
        w[idx1] = 1;
        if (deg(J[jdx1], w) > 0)          //A stricter test is to use reduced
        {
            flag1 = 1; break;
        }
    }
    if (flag1 == 1)
    {
        fatVars = fatVars + currentVar;
    }
}
// Step 3: store strings
string T_vars = string(Vars[1]);
for (int idx2 = 2; idx2 <= N; idx2++) {
    T_vars = T_vars + "," + string(Vars[idx2]);
}
string I_stored = string(I);

// Step 4: new ring F
if (size(fatVars) == 0) { print("Warning: fatVars empty."); return; }
string varsString = string(fatVars[1]);
for (int idx3 = 2; idx3 <= size(fatVars); idx3++) {
    varsString = varsString + "," + string(fatVars[idx3]);
}
string J_str = string(J);
execute("ring F = 0, (" + varsString + "), dp;");
execute("setring F;");
option(noredefine);
execute("ideal J_F = " + J_str + ";");

// Step 6: standard basis
ideal G        = std(J_F);

// Step 7: use kbase
ideal basis_JF = kbase(G);
int len_JF     = size(basis_JF);
string G_str   = string(G);
string B_str   = string(basis_JF);

// Step 8: new ring T
int m = N * len_JF;
if (m <= 0) { print("Error: fat point length < 2."); return; }
for (int idx4 = 1; idx4 <= m; idx4++) {
    T_vars = T_vars + ",a" + string(idx4);
```

```
}
execute("ring T = 0, (" + T_vars + "), dp;");
execute("setring T;");
option(noredefine);


execute("ideal I_T    = " + I_stored  + ";");
execute("ideal tmpG_T  = " + G_str     + ";");
ideal G_T        = std(tmpG_T);
execute("list basis_JF_T = (" + B_str + ");");
if (basis_JF_T[len_JF] != 1)
{
    print("Error: last basis_JF_T != 1."); return;
}

// Step 9 build arcs
list arcs;
poly expr;
int relIndex, global_idx, skip, temp, jdx2;
for (int idx5 = 1; idx5 <= N; idx5++)
{
    expr = 0;
    for (jdx2 = 1; jdx2 <= len_JF; jdx2++)
    {
        relIndex = (jdx2-1)*N + idx5;
        skip = 0;
        if (useOptList == 1)
        {
            for (temp = 1; temp <= size(optList); temp++)
            {
                if (optList[temp] == relIndex)
                {
                    skip = 1; break;
                }
            }
        }
        if (skip == 0)
        {
            global_idx = N + relIndex;
            expr = expr + var(global_idx)*basis_JF_T[len_JF - jdx2 + 1];
        }
    }
    arcs = arcs + list(expr);
}

// Step 10: substitute arcs using map
string mapStr = "map substMap = T, (";
int idx6;
for (idx6 = 1; idx6 <= N; idx6++)
{
    mapStr = mapStr + "(" + string(arcs[idx6]) + ")";
    if (idx6 < N)
    {
        mapStr = mapStr + ",";
    }
}
```

```
mapStr = mapStr + ");";
execute(mapStr);
ideal bigIdeal = substMap(I_T);

// Step 11: reduce
ideal reducedBigIdeal;
for (idx6 = 1; idx6 <= size(bigIdeal); idx6++)
{
    reducedBigIdeal = reducedBigIdeal + reduce(bigIdeal[idx6], G_T);
}

// Step 12: extract coefficients
ideal finalIdeal = 0;
poly f, term, basisElement;
poly old_f, old_term;
matrix coefMatrix;
int basisIdx, i2, kdx8, v, varIdx, shouldKeep;
string varToFind, termStr;

// Declare ov once at procedure level, before it's used
int ov = 0;

for (kdx8 = 1; kdx8 <= size(reducedBigIdeal); kdx8++)
{
    f = reducedBigIdeal[kdx8];

    for (basisIdx = 1; basisIdx < size(basis_JF_T); basisIdx++)
    {
        basisElement = basis_JF_T[basisIdx];

        coefMatrix = coef(f, basisElement);

        for (i2 = 1; i2 <= ncols(coefMatrix); i2++)
        {
            if (reduce(coefMatrix[1,i2], basisElement) == 0)
            {
                if (nrows(coefMatrix) >= 2) {
                    term = coefMatrix[2,i2];
                } else {
                    term = coefMatrix[1,i2];
                }

                // Use the pre-declared ov variable
                for (ov = 1; ov <= N; ov++) {
                    if (deg(term, ov) > 0) {
                        // Extract all coefficients including constants
                        coefMatrix = coeffs(term, var(ov));
                        poly result = 0;
                        // Process all columns in the matrix to capture every
                            coefficient
                        for (int i = 1; i <= ncols(coefMatrix); i++) {
                            result = result + coefMatrix[2,i];
                        }
                        term = result;
                    }
                }
```

```
                shouldKeep = 1;
                if (useOptList == 1)
                {
                    for (v=1;v<=size(optList);v++)
                    {
                        varIdx = optList[v];
                        varToFind = "a"+string(varIdx)+"*";
                        termStr   = string(term)+"*";
                        if (find(termStr,varToFind)!=0)
                        {
                            shouldKeep = 0;
                            break;
                        }
                    }
                }
                if (shouldKeep == 1)
                {
                    finalIdeal = finalIdeal + term;
                }
            }
        }

        for (i2=1; i2<=ncols(coefMatrix); i2++)
        {
            if (reduce(coefMatrix[1,i2], basis_JF_T[basisIdx])==0 && nrows(
                coefMatrix)>=2)
            {
                f = f - coefMatrix[2,i2]*coefMatrix[1,i2];
            }
        }
    }

    if (f!=0 && f!=1)
    {
        // Use the pre-declared ov variable again
        for (ov = 1; ov <= N; ov++) {
            if (deg(f, ov) > 0) {
                f = coef(f, var(ov))[2,1];
            }
        }

        shouldKeep = 1;
        if (useOptList==1)
        {
            for (v=1; v<=size(optList); v++)
            {
                varIdx = optList[v];
                if (varIdx <= 0) { continue; }  // Skip invalid indices

                varToFind = "a"+string(varIdx)+"*";
                termStr   = string(f)+"*";
                if (find(termStr,varToFind)!=0)
                {
                    shouldKeep = 0;
                    break;
```

```
                }
            }
        }
        if (shouldKeep == 1)
        {
            finalIdeal = finalIdeal + f;
        }
    }
}

// cleanup
poly ct;
ideal simplifiedFinal = 0;
int shouldKeep, v, i2;
string varToFind, termStr;

for (i2 = 1; i2 <= size(finalIdeal); i2++)
{
    ct = finalIdeal[i2];

    shouldKeep = 1;
    if (useOptList == 1)
    {
        for (v=1; v<=size(optList); v++)
        {
            varToFind = "a" + string(optList[v]) + "*";
            termStr   = string(ct) + "*";
            if (find(termStr, varToFind) != 0)
            {
                shouldKeep = 0;
                break;
            }
        }
    }

    // Skip terms that contain any of the original variables (first N
       variables)
    if (shouldKeep == 1)
    {
        string ctString = string(ct);
        for (int varIdx = 1; varIdx <= N; varIdx++)
        {
            string originalVar = varstr(varIdx);
            if (find(ctString, originalVar) != 0)
            {
                shouldKeep = 0;
                break;
            }
        }
    }

    if (shouldKeep == 1)
    {
        simplifiedFinal = simplifiedFinal + ct;
    }
}
```

```
finalIdeal = simplifiedFinal;

// output
print("Post-Processed Final Output:");
print(finalIdeal);

// Write the original output file (with all variables)
string origOutFile = "arc_original_output.txt";
int dummy1 = system("sh","rm -f " + origOutFile);
write(origOutFile, "ring S_full = 0, (" + T_vars + "), dp;");
write(origOutFile, "ideal A = " + string(finalIdeal) + ";", append);

// Create filtered variable list with effective variables used in final ideal
string filtered_vars = "";

// First collect all variables from N+1 to nvars(basering)
list allVars;
for (int var_idx = N+1; var_idx <= nvars(basering); var_idx++) {
    // Skip variables mentioned in optList
    int skip_var = 0;
    if (useOptList == 1) {
        for (v = 1; v <= size(optList); v++) {
            if (var_idx == N + optList[v]) {
                skip_var = 1;
                break;
            }
        }
    }

    if (skip_var == 0) {
        allVars = allVars + list(varstr(var_idx));
    }
}

// Now check every generator of finalIdeal for each variable
list usedVars;
for (int genIdx = 1; genIdx <= size(finalIdeal); genIdx++) {
    string genStr = string(finalIdeal[genIdx]);

    for (int i = 1; i <= size(allVars); i++) {
        string varName = allVars[i];

        // Check various ways the variable might appear in the polynomial
        if (find(genStr, varName + "*") != 0 || // Middle of term
            find(genStr, varName + "^") != 0 || // With power
            find(genStr, "+" + varName) != 0 || // At start after +
            find(genStr, "-" + varName) != 0 || // At start after -
            find(genStr, "*" + varName) != 0 || // After multiplication
            find(genStr, "(" + varName) != 0 || // After parenthesis
            genStr == varName ||                // Entire generator is this
                variable
            find(genStr + "+", varName + "+") != 0) { // At end before +
            // Add to used vars if not already there
            int found = 0;
            for (int j = 1; j <= size(usedVars); j++) {
                if (usedVars[j] == varName) {
```

```
                        found = 1;
                        break;
                    }
                }
                if (found == 0) {
                    usedVars = usedVars + list(varName);
                }
            }
        }
    }

    // Convert used variables list to comma-separated string
    if (size(usedVars) > 0) {
        filtered_vars = usedVars[1];
        for (int i = 2; i <= size(usedVars); i++) {
            filtered_vars = filtered_vars + "," + usedVars[i];
        }
    } else {
        // If no variables were found, just use a1 as a placeholder
        filtered_vars = "a1";
    }

    // Write filtered output file
    string outFile = "arc_output.txt";
    int dummy2 = system("sh","rm -f " + outFile);
    write(outFile, "ring S = 0, (" + filtered_vars + "), dp;");
    write(outFile, "ideal A = " + string(finalIdeal) + ";", append);
}
```

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] Federico Galetto and Nicholas Iammarino, *Computing with jets*, J. Softw. Algebra Geom., vol. 12, no. 1, 2022, pp. 43–49. (cited on page 1)

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

[4] Andrew Stout, *A sage script entitled "NFT.PY" which computes generalized jet schemes*, Version 1.0.0, June 2025. Available at https://github.com/infinity-groupoids/NFT.py (cited on page 1)

[5] 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)

[6] 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)

[7] 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)

[8] 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. (cited on page 1)