Ms. 17 April 2021

NUMERICAL SCHUBERT CALCULUS IN MACAULAY2

ANTON LEYKIN, ABRAHAM MARTÍN DEL CAMPO, FRANK SOTTILE, RAVI VAKIL, AND JAN VERSCHELDE

Abstract. The Macaulay2 package `NumericalSchubertCalculus` provides methods for the numerical computation of Schubert problems on Grassmannians. It implements both the Pieri homotopy algorithm and the Littlewood-Richardson homotopy algorithm. Each algorithm has two independent implementations in this package. One is in the scripting language of Macaulay2 using the package `NumericalAlgebraicGeometry`, and the other is in the compiled code of PHCpack.

1. Introduction

The Schubert calculus on the Grassmannian involves all problems of determining the linear subspaces of a vector space that have specified positions with respect to fixed flags of linear subspaces. The enumeration of the solution linear spaces may be solved using the Macaulay2 package `Schubert2`. Numerical Schubert calculus computes the actual solution planes to a given instance of a problem from the Schubert calculus using numerical methods, and the eponymous Macaulay2 package implements algorithms that accomplish this task.

Schubert problems arise in applications in control theory [2] and in information theory [1], and they form a rich class of geometric problems that serves as a laboratory for investigating new phenomena in enumerative geometry such as reality [11] and Galois groups [8]. The ability to compute solutions has been important in these areas. Schubert problems are also challenging to solve using standard numerical methods. This is because Schubert problems are typically not complete intersections, and even when they are, they have far fewer solutions than standard combinatorial bounds [3, p. 768].

`NumericalSchubertCalculus` has methods implementing numerical homotopy continuation algorithms that exploit explicit geometric proofs of multiplication formulas in the cohomology of a Grassmannian. The Pieri homotopy algorithm [3] is based on the geometric Pieri formula [10] and the Littlewood-Richardson homotopy algorithm [6, 12] is based on the geometric Littlewood-Richardson rule [13]. `NumericalSchubertCalculus` has two implementations of each algorithm. One is in the Macaulay2 scripting language using its `NumericalAlgebraicGeometry` [5] package and the other is compiled code using PHCpack [14].

2010 *Mathematics Subject Classification.* 14N15, 65H10.

Key words and phrases. Schubert calculus, Grassmannians, Homotopy Continuation, Littlewood-Richardson Rule, Pieri Rule.

The authors thank the American Institute of Mathematics for supporting the project through their SQuaREs program.

2. Mathematical background

A Schubert problem on a Grassmannian is a problem of determining the linear subspaces of a given dimension (its solutions) that have specified positions with respect to other fixed, but general, linear subspaces, when there are finitely many solutions. The simplest non-trivial Schubert problem asks for the two-dimensional subspaces H of $\mathbb{C}^4$ that have nontrivial intersection with each of four general two-dimensional linear subspaces $L_1,\ldots,L_4$. Replacing $\mathbb{C}^4$ by projective space, this becomes the problem of determining the lines h in $\mathbb{P}^3$ that meet four general lines $\ell_1,\ldots,\ell_4$.

This problem has two solutions. To see this, we use the classical observation that three mutually skew lines ℓ_1, ℓ_2, and ℓ_3 lie on a unique hyperboloid (Fig. 1). This hyperboloid

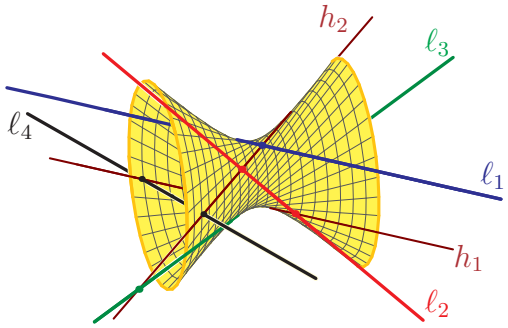

FIGURE 1. Problem of four lines

has two rulings, one contains ℓ_1, ℓ_2, and ℓ_3, and the second consists of the lines meeting these three. If the fourth line, ℓ_4, is general, then it will meet the hyperboloid in two points, and through each of these points there is a unique line in the second ruling. These two lines, h_1 and h_2, are the solutions to this instance of the problem of four lines.

Let $\mathrm{Gr}(k,n)$ be the Grassmannian of k-dimensional linear subspaces (k-planes) in $\mathbb{C}^n$. This has dimension $k(n-k)$. Indeed, a general linear subspace $H\in\mathrm{Gr}(k,n)$ is the column space of a matrix $\left(\begin{smallmatrix} X \\ I_k \end{smallmatrix}\right)$ in (reverse) column-reduced echelon form, where X is a $(n-k)\times k$ matrix, and different matrices determines a different k-planes. An incidence condition on k-planes is encoded by a *bracket*, which is an increasing sequence $\alpha\colon 1\le\alpha_1<\alpha_2<\cdots<\alpha_k\le n$ of integers. The condition is imposed by a flag F of linear spaces $F\colon F_1\subset F_2\subset\cdots\subset F_n=\mathbb{C}^n$ where $\dim F_i=i$. This pair defines a *Schubert variety*,

$$X_\alpha F \;:=\; \{H\in\mathrm{Gr}(k,n)\mid \dim H\cap F_{\alpha_i}\ge i \text{ for } i=1,\ldots,k\}\,. \tag{1}$$

Requiring that $H\in X_\alpha F$ is a *Schubert condition* on H of type α imposed by the flag F.

The Schubert variety $X_\alpha F$ has dimension $|\alpha|:=\sum_i\alpha_i-i$ and thus codimension $\|\alpha\|:=k(n-k)-|\alpha|$. A *Schubert problem* $\alpha^\bullet$ is a list $\alpha^1,\ldots,\alpha^s$ of brackets that satisfy $\sum_i\|\alpha^i\|=k(n-k)$. An *instance* of $\alpha^\bullet$ is given by a list $F^1,\ldots,F^s$ of flags, and is the geometric problem of those $H\in\mathrm{Gr}(k,n)$ that satisfy the Schubert condition α^i imposed by F^i ($H\in X_{\alpha^i}F^i$) for each i. These form the intersection

$$X_{\alpha^1}F^1\bigcap X_{\alpha^2}F^2\bigcap\cdots\bigcap X_{\alpha^s}F^s\,. \tag{2}$$

For an example, consider the problem of four lines (expressed in $\mathrm{Gr}(2,4)$), let F^i be a flag with 2-plane $F^i_2 = L_i$, for each $i = 1, \ldots, 4$. The corresponding Schubert condition is that $\dim H \cap F^i_2 \geq 1$ and $\dim H \cap F^i_4 = 2$ (as $F^i_4 = \mathbb{C}^4$), and is given by the bracket $\{2,4\}$, and so the problem of four lines is $\alpha^\bullet = (24, 24, 24, 24)$ (write 24 for the bracket $\{2,4\}$).

Kleiman [4] proved that if the flags are general, then the intersection (2) is transverse and there are finitely many solutions. The package `NumericalSchubertCalculus` implements methods to compute the solutions (2) to a given instance of a Schubert problem.

The number $d(\alpha^\bullet)$ of solutions (the points in (2)) is independent of choice of general flags, and this may be computed using algorithms in the Schubert calculus. These are implemented in the `Macaulay2` package `Schubert2`. A geometric derivation of $d(\alpha^\bullet)$ was given by the geometric Littlewood-Richardson rule in [13]. The geometric deformations of the geometric Littlewood-Richardson rule underlie the Littlewood-Richardson homotopy algorithm [6, 12] which is implemented in `NumericalSchubertCalculus`.

A Schubert condition α is *simple* if $\|\alpha\| = 1$. A Schubert problem in which all conditions except possibly two are simple is a *simple Schubert problem*. A geometric proof of the Pieri rule [10] led to the Pieri homotopy algorithm [3] for solving simple Schubert problems, and this is also implemented in `NumericalSchubertCalculus`.

3. Implementation and Syntax

The package `NumericalSchubertCalculus` has methods to compute the solutions to a given instance (2) of a Schubert problem. It includes two independent implementations of both the Littlewood-Richardson homotopy algorithm [6, 12] and the Pieri homotopy algorithm [3]. One is in the Macaulay2 scripting language and the other is compiled code. These implementations have slightly different capabilities and input syntax, with the package providing some interoperability. We briefly describe the input and output syntax of the primary methods, and their capabilities.

In both, brackets are represented by lists so that `{3, 5, 6}` is a simple Schubert condition when $k = 3$ and $n = 6$. Flags are represented by invertible $n \times n$ matrices, with F_i the span of the first i columns. An element H of $\mathrm{Gr}(k,n)$ is represented by an $n \times k$ matrix whose column span is H.

3.1. **Scripted methods.** These compute the solutions to a given instance of a Schubert problem. An instance of a Schubert problem $\texttt{SchubProb} = \{\alpha^1, \ldots, \alpha^s\}$ on $\mathrm{Gr}(k,n)$ is represented by a list of pairs

$$\texttt{SchProbInst} \;=\; \{\, \{\alpha^1, F^1\}\,,\, \{\alpha^2, F^2\}\,, \ldots\,,\, \{\alpha^s, F^s\}\,\}\,, \tag{3}$$

where each pair $\{\alpha^i, F^i\}$ consists of a bracket α^i and a flag F^i. The parameters k, n are implicit in the data (k is the length of the bracket and n is the size of the matrix).

Given an instance (3) of a Schubert problem on $\mathrm{Gr}(k,n)$, the solutions are computed with the method `solveSchubertProblem`. A typical call is

```
Solns = solveSchubertProblem(SchProbInst, k, n).
```

(The parameters k, n are included for internal verification that `SchProbInst` is in fact an instance of a Schubert problem on $\mathrm{Gr}(k,n)$.) This returns the solutions as a list

`Solns` of $n \times k$ matrices $H_1, \ldots, H_d$, each of whose column span is a k-plane solving the instance represented by `SchProbInst` that is, it is a point in the intersection (2). As the computation operates in local coordinates for the Grassmannian, it presupposes that the flags are sufficiently general so that the solutions lie in the local coordinates.

The method `randomSchubertProblemInstance` returns a random instance of a Schubert problem `SchubProb` $= \{\alpha^1, \ldots, \alpha^s\}$ on $\mathrm{Gr}(k,n)$,

```
RandSchProbInst = randomSchubertProblemInstance(SchubProb, k, n).
```

This gives an instance (3) of `SchubProb` in which the flags F^i are random complex matrices. Using the output `RandSchProbInst` as input for `solveSchubertProblem` gives solutions to the random instance, and not a user-chosen instance. `NumericalSchubertCalculus` provides a method, `changeFlags`, based on the parameter or cheater's homotopy [7, 9] that, given the solutions `Solns` to a particular instance of a Schubert problem `SchubProb`, represented by a list of flags `F`, computes the solutions to a different instance represented by another list of flags `myF`. A typical call is

```
mySols = changeFlags(Solns, SchubProb'F'myF),
```

where `SchubProb'F'myF` is a triple {`SchubProb, F, myF`}

If `SchubProb` is a simple Schubert problem, then the alternative Pieri homotopy algorithm is available, and it may be used to compute the solutions with the call

```
Solns = solveSimpleSchubert(SchubProb, k, n).
```

This requires that all conditions in the instance `SchubProb` are simple, except possibly the first two. The previous discussion of a random versus a user-provided instance and the method `changeFlags` for interpolating between them also applies here.

One exported method is `checkIncidenceSolution`, which is a Boolean-valued function that may be used to check if a given k-plane H satisfies an instance of a Schubert problem. A typical call is

```
checkIncidenceSolution(H, SchProbInst).
```

Here, `H` is a $n \times k$ matrix and `SchProbInst` is a list of pairs (3).

The exported method `LRnumber` computes the number of solutions to a given Schubert problem `SchubProb`. A typical call is

```
d = LRnumber(SchubProb, k, n),
```

where `SchubProb` is a list of brackets that constitute a Schubert problem on $\mathrm{Gr}(k,n)$ and the output `d` is an integer.

While it is straightforward (1) to encode a Schubert condition in $\mathrm{Gr}(k,n)$ as a bracket $\alpha\colon 1 \le \alpha_1 < \cdots < \alpha_k \le n$, a common equivalent encoding is a partition, which is a weakly decreasing sequence $\lambda\colon n-k \ge \lambda_1 \ge \cdots \ge \lambda_k \ge 0$. For example, `Schubert2` uses partitions. A bracket α and its corresponding partition λ satisfy

$$\alpha_i - i \;+\; \lambda_i \;=\; n-k \quad \text{for} \;\; i = 1, \ldots, k\,.$$

In this case, $|\lambda| = \lambda_1 + \cdots + \lambda_k$ equals the codimension $\|\alpha\|$. The exported methods `bracket2partition` and `partition2bracket` translate between the two notations. The

input for methods described so far allow a Schubert problem to be expressed as either a list of brackets or a list of partitions.

3.2. **Interface to PHCpack.** The corresponding routines that interface with the compiled implementations in PHCpack have minor differences in syntax and capability to those in Section 3.1. The main difference is that these methods compute solutions to a random instance of the given Schubert problem that is generated when the method is called. The output includes the instance that was used for the computation. To obtain solutions to a user's instance requires using `changeFlags`.

The method `LRtriple` implements the geometric Littlewood-Richardson rule. It encodes a Schubert problem slightly differently than the method `solveSchubertProblem`; rather than a list of brackets, it takes a matrix whose rows have the form `{m, b}` where `b` is a bracket (increasing sequence of length k) and `m` is the *multiplicity* of that bracket in the Schubert problem (how often it appears). The method `NSC2phc` takes as input a list of brackets and its output is a matrix encoding the Schubert problem. A typical call is

$$\texttt{M} \;=\; \texttt{NSC2phc}(\texttt{SchubProb}, \texttt{k}, \texttt{n})\,.$$

Given a matrix `M` encoding a Schubert problem on $\mathrm{Gr}(k,n)$, a typical call of `LRtriple` is

$$(\texttt{F}, \texttt{P}, \texttt{S}) \;=\; \texttt{LRtriple}(\texttt{n}, \texttt{M})\,.$$

The output is a triple of strings with `F` describing the flags and the solutions, `P` is the polynomial system solved in local coordinates, and `S` are the solutions in local coordinates and information about the computation (see the documentation). The method `parseTriplet` transforms these strings into Macaulay2 objects for further processing. A typical call is

$$(\texttt{R}, \texttt{pols}, \texttt{sols}, \texttt{fixedFlags}, \texttt{movedFlag}, \texttt{solutionPlanes}) \;=\; \texttt{parseTriplet}(\texttt{F}, \texttt{P}, \texttt{S})\,.$$

This is a list of Macaulay2 objects which enable further processing of the computed solutions (again, see the documentation).

The routine `LRrule` is a PHCpack implementation of a method to compute the number of solutions to a given Schubert problem using the geometric Littlewood-Richardson rule. A typical call is

$$\texttt{s} \;=\; \texttt{LRrule}(\texttt{n}, \texttt{M})\,,$$

where `n` is the ambient dimension, and `M` is a matrix encoding the Schubert problem. The output `s` is a string encoding a product in the cohomology ring. For problem of four lines, this is `[ 2 4 ]^4 = +2[1 2]`.

3.3. **A note on numerics.** `NumericalSchubertCalculus` implements highly recursive algorithms based on numerical homotopy continuation and path-tracking. Because of the recursion, when running the scripted algorithms of Section 3.1, it is advisable to increase `Macaulay2`'s limit on the allowed depth of a recursion, which is controlled by the `recursionLimit` variable. As the algorithms are numerical, at times the path-tracking will fail. One work-around is to set the seed for the random number generator. For the scripted algorithms of Section 3.1, this may be done by changing the variable `setRandomSeed`, and for the compiled algorithms in Section 3.2, this is done with the option `luckySeed`.

Numerical computation in `NumericalSchubertCalculus` takes place in systems of local Stiefel coordinates for subvarieties of $\mathrm{Gr}(k,n)$. While details are explained in [6, §2.2], we note the following consequence: When computing solutions to a Schubert problem $\alpha^1,\ldots,\alpha^s$, the maximum number of coordinates (the ambient dimension of the computation) is $k(n-k)-\|\alpha^1\|-\|\alpha^2\|$. The computational heuristic that using fewer coordinates improves performance holds in practice with our software. Both our implementations of the Littlewood-Richardson homotopy, `solveSchubertProblem` and `LRtriple`, are faster and more numerically stable when the conditions satisfy

$$\min\{\|\alpha^1\|,\|\alpha^2\|\} \;\geq\; \max\{\|\alpha^3\|,\ldots,\|\alpha^s\|\}\,.$$

Consequently, a user is advised to sort their Schubert conditions so that the largest two are first.

4. Examples

We close with some examples that illustrate our software. The problem of four lines in $\mathrm{Gr}(2,4)$ is given by four brackets $\{\{2,4\},\{2,4\},\{2,4\},\{2,4\}\}$.

```
i4 : SchubProb = randomSchubertProblemInstance({{2,4},{2,4},{2,4},{2,4}}, 2,4)

o4 = {({2, 4}, | -.114281-.993448ii -.0168841-.999857ii -.0141397-.9999ii....
               | -.788054+.615606ii -.319468+.947597ii  .980731-.195363ii....
               | .300518-.953776ii  .0796792-.996821ii  .624656-.7809ii  ....
               | .676179+.736737ii  -.886912+.461938ii  -.237463+.971397i....
     ------------------------------------------------------------------------....

o4 : List

i5 : solveSchubertProblem(SchubProb,2,4)

o5 = {| -.140237+1.14381ii -.378754+.204742ii |, | .0466437-.65223ii    -....
      | 1.57394-.065106ii  .0818881+.152422ii |  | -.0559878+.7121ii    1....
      | -.906612+.910475ii .0437461-.383392ii |  | -.00398277-.653482ii -....
      | -2.04662-1.34516ii .0997269+.124121ii |  | -1.14257+.21797ii    1....

o5 : List
```

The output `o5` consists of two 4×2 matrices whose column spaces solve the given instance of this Schubert problem. Its output is nearly instantaneous. Here is a significantly larger Schubert problem in $\mathrm{Gr}(4,8)$ with 1530 solutions, computed using `LRtriple`:

```
i1 : M = matrix {{2, 3,5,7,8 },{1, 3,6,7,8}, {8, 4,6,7,8}};

               3        5
o1 : Matrix ZZ  <--- ZZ

i2 : result = LRtriple(8, M);
PHCv2.4.82 released 2020-11-30
```

(Some output is suppressed.)

```
i3 : L= lines(result_2);

i4 : L_1

o4 = 1530 10

i5 : L_(#L-10)

o5 = User time in seconds was                    336.834157000 =  0h 5m36s834ms
```

We now compute the simple Schubert problem $\{3,5,6\}^9$ in $\mathrm{Gr}(3,6)$ with 42 solutions in four different ways. The bracket $\{3,5,6\}$ corresponds to the partition $\{1\}$.

```
i4 : k=3, n=6;

i5 : conds = {{1},{1},{1},{1},{1},{1},{1},{1},{1}};

i6 : LRnumber(conds,k,n)

o6 = 42

i7 : SchPblm = randomSchubertProblemInstance(conds,k,n);

i8 : time S = solveSchubertProblem(SchPblm,k,n);
     -- used 96.698 seconds

i9 : #S

o9 = 42

i10 : time S = solveSimpleSchubert(SchPblm,k,n);
     -- used 0.877903 seconds

i11 : #S

o11 = 42

i12 : M = NSC2phc(conds,k,n)

o12 = | 9 3 5 6 |

               1        4
o12 : Matrix ZZ  <--- ZZ

i13 : s = LRrule(n,M)
PHCv2.4.82 released 2020-11-30

writing data to file /tmp/M2-18834-0/2PHCinput
```

```
running phc -e, writing output to /tmp/M2-18834-0/3PHCoutput
opening output file /tmp/M2-18834-0/3PHCoutput

o13 = [ 3 5 6 ]^9 = +42[1 2 3]

i14 : time (F,P,S) = LRtriple(n,M);
PHCv2.4.82 released 2020-11-30
```

(Some output is suppressed.)

```
     -- used 0.122594 seconds

i15 : (lines(S))_1

o15 = 42 7

i16 : time (ipt, otp) = PieriHomotopies(3,3);
     -- used 0.0915026 seconds

i17 : ipt

o17 = {| -.408248           .233797-.146195ii   -.339562+.0273021ii  |, ....
       | .403083-.0647356ii -.361496+.0525928ii .170936-.210227ii    |  ....
       | -.35011+.209975ii  .181004-.236536ii   -.245246+.0877148ii  |  ....
       | -.332084+.237459ii -.232994+.556345ii  .121023+.101895ii    |  ....
       | .403492+.0621347ii .524032-.051926ii   .0282015-.00137244ii |  ....
       | .168565-.371823ii  .119146+.21564ii    -.0367667+.845884ii  |  ....

o17 : List

i18 : #otp

o18 = 42
```

References

1. G. Bresler, D. Cartwright, and D. Tse, Feasibility of interference alignment for the MIMO interference channel, IEEE Trans. Inform. Theory **60** (2014), no. 9, 5573–5586.
2. C.I. Byrnes, Pole assignment by output feedback, Three Decades of Mathematical Systems Theory (H. Nijmeijer and J. M. Schumacher, eds.), Lecture Notes in Control and Inform. Sci., vol. 135, Springer-Verlag, Berlin, 1989, pp. 31–78.
3. B. Huber, F. Sottile, and B. Sturmfels, Numerical Schubert calculus, Journal of Symbolic Computation **26** (1998), no. 6, 767 – 788.
4. S.L. Kleiman, The transversality of a general translate, Compositio Mathematica **28** (1974), 287–297.
5. A. Leykin, Numerical Algebraic Geometry, The Journal of Software for Algebra and Geometry **3** (2011), 5–10, Available at `http://j-sag.org/volume3.html`.
6. A. Leykin, A. Martín del Campo, F. Sottile, R. Vakil, and J. Verschelde, Numerical Schubert Calculus, Mathematics of Computation **90** (2021), 1407–1433.
7. T. Y. Li, Tim Sauer, and J. A. Yorke, The cheater's homotopy: an efficient procedure for solving systems of polynomial equations, SIAM Journal on Numerical Analysis **26** (1989), no. 5, 1241–1251.

8. A. Martín del Campo and F. Sottile, Experimentation in the Schubert calculus, Schubert Calculus—Osaka 2012 (H. Naruse, T. Ikeda, M. Masuda, and T. Tanisaki, eds.), Advanced Studies in Pure Mathematics, vol. 71, Mathematical Society of Japan, 2016.
9. A. P. Morgan and A. J. Sommese, Coefficient-parameter polynomial continuation, Appl. Math. Comput. **29** (1989), no. 2, part II, 123–160.
10. F. Sottile, Pieri's formula via explicit rational equivalence, Canadian Journal of Mathematics. Journal Canadien de Mathématiques **49** (1997), no. 6, 1281–1298.
11. ———, Real solutions to equations from geometry, University Lecture Series, vol. 57, American Mathematical Society, Providence, RI, 2011.
12. F. Sottile, R. Vakil, and J. Verschelde, Solving Schubert problems with Littlewood-Richardson homotopies, ISSAC 2010—Proceedings of the 2010 International Symposium on Symbolic and Algebraic Computation, ACM, New York, 2010, pp. 179–186.
13. R. Vakil, A geometric Littlewood-Richardson rule, Annals of Mathematics. Second Series **164** (2006), no. 2, 371–421, Appendix A written with A. Knutson.
14. J. Verschelde, Algorithm 795: PHCpack: A general-purpose solver for polynomial systems by homotopy continuation, ACM Transactions on Mathematical Software **25** (1999), no. 2, 251–276.

School of Mathematics, Georgia Institute of Technology, 686 Cherry Street, Atlanta, GA 30332-0160, USA
Email address: leykin@math.gatech.edu
URL: http://people.math.gatech.edu/~aleykin3/

Abraham Martín del Campo, Centro de Investigación en Matemáticas, A.C., Jalisco S/N, Col. Valenciana, Guanajuato, Gto. México
Email address: abraham.mc@cimat.mx
URL: http://personal.cimat.mx:8181/~abraham.mc/Home.html

Frank Sottile, Department of Mathematics, Texas A&M University, College Station, Texas 77843, USA
Email address: sottile@math.tamu.edu
URL: www.math.tamu.edu/~sottile

Ravi Vakil, Department of Mathematics, Stanford University, Stanford, CA 94305 USA
Email address: rvakil@stanford.edu
URL: http://math.stanford.edu/~vakil

Jan Verschelde, Dept of Math, Stat, and CS, University of Illinois at Chicago, 851 South Morgan (M/C 249), Chicago, IL 60607 USA
Email address: jan@math.uic.edu
URL: http://www.math.uic.edu/~jan