COLLABORATIVE RESEARCH: SI2-SSI: SAGEMATH CROSS PRODUCT: Community Run Open Source Software Providing Research Opportunities for Developers, Users, Contributors, and Teachers

SageMath [217], or Sage for short, is a free **open source** general purpose mathematical software system, initiated under the leadership of William Stein (University of Washington), that has developed explosively within the last ten years among research mathematicians. Stein's primary motivational goal was to create software that is **efficient**, **open source**, **comprehensive**, **well-documented**, **extensible**, and **free** [216]. Sage's development has reached a point where it has the potential to become a standard software package that is widely used throughout STEM disciplines. If funded, the proposed project will allow us to make major progress towards this goal.

Sage is similar to Maple [150], Mathematica [237], Magma [43], and MATLAB [225] (via third-party packages like SciPy [105]), but uses the popular Python language [175] both as an implementation and as a surface language. This not only ensures easy access to help with programming [75, 146, 174], but also provides developers with **transferable skills** as they work with a computer language that is ubiquitous in the software industry. Furthermore, Sage has gained strong momentum in the mathematics community far beyond its initial focus in number theory, in particular in the field of combinatorics where the capabilities and **functionalities** of Sage are much **deeper** than those of commercial packages.

Sage is **community-run**, meaning that it is developed and organized by users for users. The development is driven by research and teaching needs and has a variety of stakeholders: e.g., developers, users, contributors, researchers, and teachers. Sage has over 180 developers [171] (more than 140 currently active [165]). Additionally, Sage combines dozens of open source packages, thus benefiting from development by an even larger community comprised of a wide range of mathematicians, physicists, and engineers. With the development of SageMathCloud [218], thematic tutorials [224], and other systems such as SageTex [72] and FindStat [222, 38], Sage is becoming more appealing, more accessible, and easier to use for **research opportunities** and **teaching purposes**. SageMathCloud has more than 30,000 active users. There have been more than 300 updated releases of Sage, with both precompiled binary archives for a variety of platforms and Sage's source code available [217], being downloaded over 6,500 times per month on average [171].

In this proposed project, our main objectives are to:

(1) Deepen Sage functionality in fundamental areas of mathematics amenable to computation;
(2) Lower Sage's barriers to entry for researchers, educators, students, & industry professionals and enhance it as a tool for STEM education;
(3) Engage with and broaden the larger Sage development community; and
(4) Improve Sage's interface with the wider open source software community.

Open source projects need a critical mass of people to sustain themselves (e.g., Wikipedia or the widely-used Online Encyclopedia of Integer Sequences (OEIS) [212]). Sage has made great progress, but has not yet reached this point. It is imperative to train the next generation in Sage to reach this critical mass. The above objectives **incentivize a broad scientific community** to devote their time using, developing, or interfacing with Sage by providing tools immediately relevant to their educational and research endeavors.

We achieve these goals with the following key activities:

- We **build** on existing functionality in combinatorics to develop new code in three neighboring areas: statistical mechanics, representation theory, and optimization. (Section 2)
- We **leverage** new pedagogical tools to produce modules for a wide variety of coursework in mathematics from calculus to graduate courses. (Section 3)

- We **foster** development in additional areas of mathematics through SAGE Days workshops, sponsored small-group coding sprints as follow-ups, mentorship of young people (particularly those in underrepresented groups), and collaboration with OpenDreamKit to provide international and interdisciplinary distribution of our products. (Section 4)
- We **deliver** structural improvements to both SAGE and the wider open source community, reducing fragmentation of existing software communities and translating SAGE's best practices to a wider community of software systems. (Section 5)

1. Why Sage is our choice of software for this project

1.1. **SAGE is designed for science.** Two bedrock principles of basic science, mathematics in particular, are **transparency** and **reproducibility**. Computational software is becoming pervasive as a crucial research tool for constructing examples, verifying conjectures, and running large-but-finite computations or case-by-case proofs (the four-color problem being a famous example). However, proprietary software lacks transparency; this makes the code built on top of it far more difficult to maintain as the base software upgrades, rendering computations irreproducible. Because of this, the use of **open source** software, such as SAGE, which is released under the GPL version 2+, is essential for the free exchange of ideas and the ability to detect errors.

Moreover, SAGE is designed the way modern scientists and mathematicians think, is optimized for the **understanding of mathematics** and provides a programming environment that closely mimics the structure of mathematics. SAGE developers drew on the development work of earlier mathematical software packages (AXIOM, MAGMA, ALDOR, MUPAD, and FRICAS). In addition to supporting **object-oriented programming**, SAGE uses **mathematical categories** (e.g., of groups, of rings, or of fields) and **axioms** (e.g., associativity, existence of inverse) as a design pattern to dynamically construct its complicated class hierarchy from duplication-free semantic information. In SAGE, this infrastructure is built on top of (and not into) the standard PYTHON class hierarchy framework and does not require changing the language itself. By building this structure, SAGE is able to organize mathematical objects from a wide variety of settings, while still allowing them to interact when necessary; see [69].

1.2. **SAGE is an integrating force.** Since SAGE is built upon a number of open source software packages, in addition to its own code base in PYTHON and CYTHON [36], it can provide a powerful engine for computations utilizing existing state-of-the-art libraries rather than developing the same features from scratch. Further, the output of computations from these various highly specialized and optimized packages can all be processed by the user from within a **common platform**. For example, to compute a set of matrices with entries in a finite field of size 2^8 that satisfy some polynomial equation, one uses a number theory library like NTL [210]. If this set is closed under matrix multiplication (i.e., forms a group), then to compute the character table one needs GAP. In contrast, SAGE's code base incorporates both of these packages and thus requires the user to write only a few lines of code. For a full list of standard packages accessible through SAGE, see [213]; we highlight here a few standard packages used by a broad collection of scientific researchers, whose accessibility and usefulness is significantly enhanced by their inclusion in SAGE.

Related and incorporated software: R [176] for statistics; GAP [88] for group theory; SINGULAR [67] for polynomials and commutative algebra; MPIR [96], MPFR [80, 154], and MPFI [178] for multiprecision arithmetic; MAXIMA [147], PYNAC [219], and SYMPY [160] for symbolic computations; ATLAS [235] for matrix computations; NTL [210], FLINT [95], and PARI [168] for number theory; PARMA POLYHEDRA LIBRARY [15] (PPL for short) for exact polyhedral computations and mixed-integer linear optimization; GLPK [144] for large-scale numerical mixed-integer linear optimization; CVXOPT [7] for convex optimization; CDDLIB [85] for polyhedral computations; SYMMETRICA [131] for symmetric functions.

Not only does SAGE incorporate over 100 different open source software components, but its developer community of 180 people is already larger than most. For comparison, MAXIMA has 41 developers (18 currently active) [226], GAP has 54 (19 currently active) [87], and SINGULAR has 95 developers [211]. This incorporation also provides these other software communities with a wider audience and potential upstream improvements. In this way, SAGE plays an integrating role similar to that of Linux distributions. For the large number of included upstream packages SAGE provides a degree of **quality control** by building and testing them on several platforms. While many of these packages also appear in comprehensive Linux distributions (e.g., Debian), the testing provided by SAGE through the tight mathematical integration is more rigorous.

One particularly noteworthy package is CYTHON, a programming language that gives C-like performance with PYTHON syntax (making code nearly interchangeable and very easy for PYTHON programmers to learn) and provides simple hooks for C/C++ code to be called from PYTHON code. In fact, CYTHON began in 2007 as a fork of PYREX [74] for SAGE, and has since grown to be an independent project, very popular among scientific communities that use PYTHON.

In addition to the standard packages above, there are a number of additional optional packages that can be installed as part of SAGE such as GAMBIT [149], COXETER3 [73], CHOMP [170, 108], 4TI2 [1], and LATTE INTEGRALE [223], which do computations in game theory, Coxeter groups, homology, polyhedral geometry, and lattice point enumeration. SAGE provides bindings to C/C++ (as mentioned above), FORTRAN, and LISP code, as many libraries are written in these languages. SAGE also provides an interface to a number of proprietary software packages such as MATHEMATICA, MAGMA, and MAPLE and proprietary mixed-integer linear programming solvers, e.g., CPLEX and GUROBI, thereby providing **one common point of access** and **one common language** for all of these programs. In addition, SAGE provides interfaces to two web databases highlighted in the Notices of the American Mathematical Society [41]: OEIS [212] and FINDSTAT [222], the combinatorial statistics database, which uses SAGE as a backend computational tool. As another tool, any computational results of SAGE can also be embedded directly into LATEX files via SAGETEX [72].

1.3. **SAGE code is sustainable.** Before SAGE, there were several mathematical software ecosystems where a researcher or educator driven by the need for computer experimentation would write a tool for their purposes. Often times such code, especially those written for proprietary software, contain little to no unit tests or documentation. The lack of documentation makes the code difficult to use, e.g., there are hidden assumptions about the input. Absence of unit tests can make the code fragile, e.g., a bug fix in one place can break other functionality. Furthermore, third-party packages for proprietary software often break with new releases of the underlying software.

In contrast, code to be included in SAGE must be fully **documented** and pass **doctests** for every subsequent release. This also ensures that code in SAGE will remain valid with each new release, preventing the current common frustration mentioned above. Further, in SAGE, any accepted code that is removed or whose application programming interface (API) changes must first be deprecated for a least a year. This gives users ample warning of changes that they may need to make in their personal code to accommodate changes in SAGE and the ability to request that a feature be reinstated.

1.4. **The SAGE community is sustainable.** With the addition of SAGE into the computer algebra software ecosystem, many first time developers are **choosing to code** in SAGE for all of the reasons outlined in Sections 1.1–1.3. One further compelling reason for SAGE is that both its code base and interfaces are written in PYTHON, thus the user/developer does not need to learn two separate languages to use and develop in SAGE. PYTHON is a standard language known by millions of people, used especially in the scientific computing community. Many of the graduate students and postdocs who have been trained developing for SAGE later found **employment in industry**, working for such companies as YouTube/Google, Spiceworks, and Epic.

The large user base of SAGE has maintained its size consistently over the last several years, but also has strong potential to grow with even more exposure. It is the strength of this user base that ultimately will sustain this project beyond the lifetime of this NSF grant. The SAGE mantra is "coding done by users for users"; by achieving **critical mass** (a realistic goal to achieve in the next five years through the programs outlined in this proposal), we will ensure that there continues to be several developers familiar with any given feature and that the code base is maintained and strengthened for years to come.

1.5. **SAGE development model.** Let us describe the typical development **work flow** from inception to **proof-of-concept design** to final integration into SAGE. This work flow serves to **enforce best practices** and will apply to the projects described in Section 2.

(1) Mathematical research, teaching needs, or industry suggest a particular **new feature** or a **bug** is discovered in the current code.
(2) Various contributors discuss **designs** and **algorithms** for the feature or bug fix at SAGE Days, in person, or electronically, and then collaboratively write some stand-alone code in the SAGEMATHCLOUD as proof-of-concept.
(3) Developers discuss where in the code base this feature should be integrated. Once the location is determined, they transform the stand-alone code into a GIT **branch** on top of the current SAGE code with proper **documentation** and a **test suite**. This provides examples and ensures that possible conflicts with future changes can be detected and resolved.
(4) The GIT branch is attached to a **trac ticket** on the SAGE trac server [227].
(5) One or more developers review the code.
(6) When the code receives positive reviews, the SAGE release manager integrates the code into the **next release** of SAGE.

A new stable version of SAGE is publicly released on average every 1–2 months with the latest features. This allows SAGE to link with the latest innovations in mathematical computations, keeping it close to state-of-the-art (but with no need to purchase upgrades).

2. DEEPENING CORE FUNCTIONALITIES

2.1. **Statistical mechanics and exactly solvable models.** Statistical mechanics originated in physics as a means of understanding global phenomena from all possible local interactions of particles, each of which produces an "admissible state" of the model. The field has grown to include important connections to probability, dynamical systems, combinatorics, special functions, and now gauge theory as showcased in Costello and Witten's talks at *Strings 2016* [236]. We focus on the quantum mechanical case when the admissible states are a discrete family. A central question in the subject is the evaluation of generating functions on the set of admissible states, known as partition functions. If the long-term behavior or exact formulas for the partition function are known, then the model is termed "exactly solvable" [32, 103]. From the physics point of view, these partition functions provide the total probability measure essential to determining the probability of any given observable and detecting phase transitions. We first provide a timeline for implementing a complete tool kit for exploring solvable lattice models. (Brubaker and Scrimshaw)

Year 1: Enlarge and optimize the list of solvable two-dimensional lattice models in SAGE. Code for the six-vertex model by Scrimshaw is in SAGE, but could have greater flexibility in its input of Boltzmann weights. The eight-vertex, Kagomé, and hard hexagon models as outlined in [32] will be implemented using similar code. (with Schilling)

Years 1–2: Implement a Yang-Baxter equation (a.k.a. star-triangle relation) solver for interactive solutions to these non-linear equations — a key tool in the solvability of the model. Proof-of-concept code by Brubaker and D. Bump was used for the six-vertex model and certain generalizations in [44].

Year 2: Implement approximation methods for partition functions with arbitrary Boltzmann weights using statistical sampling and power series expansion. Develop a graphical interface for plotting results to determine phase transitions of the model.
Year 3: Add methods of the Kyoto school on algebraic analysis of difference equations arising from solvable lattice models [102]; in particular, determinantal methods and Wick's theorem for evaluation of tau functions in the boson-fermion correspondence as in [241].
Years 4–5: Generalize the above tools to higher dimensional models, e.g., implementing three-dimensional lattice models and a tetrahedral equation solver as in [33].

From the algebraist's point of view, a large class of functions important in representation theory, combinatorics, and number theory arise as partition functions of such models. We can take advantage of and build upon the deep reservoir of algebraic combinatorics in Sage to study these partition functions and related dynamical systems from this new perspective. The algebraic and combinatorial objects arising in formulas for these partition functions allow us to explore the significance behind, or reason for, the positive expansion of Macdonald polynomials in bases of symmetric functions [129, 93] or Laurent expansions of cluster variables [77, 53].
Years 1–2: Provide dictionaries between admissible states of lattice models and algebraic combinatorial models (tableaux, plane partitions, alternating sign matrices, etc.) and statistics on both (e.g., charge, energy) [206, 209, 127, 155]. (Schilling, Striker, Dilks, UCD NSF postdoc Gillespie)
Years 1–2: Provide dictionaries between six-vertex models, T-systems and cluster algebras and implement associated combinatorial formulas as in [70]. (Recent UMN Ph.D. graduate Gunawan, Musiker, Scrimshaw, UMN Ph.D student Strasser)
Year 3: Interpretations for Macdonald polynomials as partition functions of lattice models have been given in [54]. We will implement their model and use the dictionaries to explore expansions in other bases of symmetric functions. (Brubaker, Schilling, Scrimshaw)
Years 3–4: Using plane partitions infrastructure and code of V. Pons, implement dual stable Grothendieck functions [125] and their non-dual and non-stable analogues. (Grinberg, Musiker).
Years 4–5: Implement an ultradiscrete version of the Korteweg-de Vries (KdV) equation known as box-ball systems [99], based upon code written by Scrimshaw. (Grinberg, Musiker, Scrimshaw)
Year 5: Crystal bases first arose from statistical mechanics and are related to five- and six-vertex models. Recent research [44] suggests new connections between their partition functions and modules of super Lie algebras. The affine crystals associated to these modules will be implemented. This is connected to goals in Section 2.2. (Brubaker, Schilling, Scrimshaw)

Lattice models can be viewed as discrete time evolution of one-dimensional systems. This has led to research on discrete dynamical systems arising from the algebraic and combinatorial objects above. The workshop on "Dynamical Algebraic Combinatorics" at the American Institute of Mathematics [238], for which Roby and Striker were organizers, made significant progress in understanding such systems, such as toggle systems and cluster algebras, by using Sage code to uncover relationships to existing mathematics [71]. Dilks, Roby, Striker, and students plan to implement the following to enable expanded research of these systems:
Years 1–2: Implement and enhance discrete dynamical systems, including toggle systems [221], Bulgarian solitaire, and Suter rotation [181].
Years 3–4: Implement general infrastructure for finding orbit-averages of statistics in discrete, piecewise-linear, and birational dynamical systems [181]. This will enable further research on the homomesy phenomenon [173] and the Razumov-Stroganov correspondences [55, 56, 177, 220].
Year 5: Implement code to systematically study orbit structure and the degree of resonance exhibited by a discrete dynamical system [71].

For toggling of posets, an algebraic geometric analogue known as birational rowmotion was implemented in Sage by Grinberg. Starting with work at AIM [238], Grinberg and Glick (AIM

participant) discovered a connection to cluster algebras, and Musiker and Roby realized combinatorial formulas, for such dynamics in the case of rectangular posets. This has inspired us to think about implementations of non-commutative versions of these dynamics in a number of contexts.
Years 1–2: Implement a non-commutative version of birational rowmotion that relates to a conjecture of Kontsevich [115]. Calculations are difficult even in small cases so a computer implementation is paramount. (Grinberg, Musiker, Roby)
Years 2–3: There are also non-commutative versions of the theory of cluster algebras, one such theory being quantum cluster algebras [37]. An implementation of quantum cluster algebras into SAGE would lead to research in even more cases and lend applications to hyperbolic geometry and knot theory through the quantum Teichmüller space [76]. (Grinberg, Musiker).
Year 3: Non-commutative cluster algebras and quantum Teichmüller spaces also naturally arise in continuous dynamical systems known as cluster integrable systems [81, 90]. An implementation of cluster algebras from toric diagrams could start by porting over code that physicists such as Francis Lam, Sebastian Franco, and Yang-Hui He have written in MATHEMATICA. (Musiker)
Years 3–5: In addition to the aforementioned T-systems and cluster integrable systems, there are also Y-systems that originated from the thermodynamical Bethe Ansatz (as in Zamolodchikov Periodicity [240]), and were connected to cluster algebras by Fomin and Zelevinsky [78]. We will implement Y-systems and their non-commutative analogues (e.g., quantum dilogarithms) for cluster algebras by extending Scrimshaw's Q-systems code. (Musiker, Scrimshaw)

2.2. **Representations of algebras.** Representation theory of algebras is fundamental in understanding structures arising from many different areas in mathematics and physics, including geometry, number theory, combinatorics, quantum physics and string theory. Let us illustrate the interplay between representation theory and computation. In 1990, Kass, Moody, Slansky and Patera published a two-volume work [111] on representations of affine Lie algebras [107], including tables of weight multiplicities of their representations and branching rules. In particular, in the second volume, which consists of these tables, they write in the introduction:

> We present a vast quantity of numerical data in tabular form .... It would indeed be gratifying if these tables were to appear to the scientists of 2040 as obsolete as the dust-gathering compilations of transcendental functions appear for us today because of their availability on every pocket calculator.

These computations can now be done in SAGE thanks to a previous OCI SAGE grant (see Section 7), and the implemented package is helping researchers explore intriguing connections between weight multiplicities of affine Lie algebras and modular forms in number theory. However, there are no software implementations of these Lie algebras nor their representations; code in GAP merely handles the finite-dimensional case. Thus, many of the Lie algebras interesting to physicists (e.g., infinite-dimensional Heisenberg, Virasoro) are not available. Likewise, Lie super-algebras, i.e., Lie algebras with even and odd parts, naturally appearing in statistical mechanics and conformal field theory, also lack implementations. Furthermore, GAP does not have any specialized linear algebra code, and SAGE's more optimized linear algebra code allows for the construction of much larger examples in the finite-dimensional case. By providing an implementation of these Lie algebras, we afford researchers the opportunity to explore the structure and properties of their representations such as invariant subspaces and branching rules.

Additionally, we can give a combinatorial interpretation of certain Lie algebra representations by using Drinfeld–Jimbo quantum groups and crystal bases [143, 110, 109]. Crystal bases are also known to be tightly intertwined with the representation theory of Hecke algebras [8, 128]. A certain subalgebra of a quantum group has an incarnation as a category using so-called Khovanov–Lauda–Rouquier (KLR) algebras [112, 182, 183] in a way that respects its crystal structure [130], which has led to important applications, e.g., [49, 114, 123]. Hence, we give another avenue to examine the

representation theory of Lie algebras and quantum groups by providing the first implementation, as far as we are aware, of KLR algebras.

The description of our planned implementations is given below.

Year 1: The Fock space representation for the full general linear quantum group based on code written by Scrimshaw will be finalized (Grinberg, Lee, Schilling, Scrimshaw).

Years 1–2: Based on previous code by Scrimshaw, a generic framework for Lie (super-)algebras in SAGE will be implemented. This will include implementations of the infinite-dimensional Heisenberg and Virasoro algebras, along with an interface for GAP and LiE [231]. (Grinberg, Lee, Scrimshaw)

Years 1–2: Develop an optional SAGE package that includes GAP procedures for category theory that can interface with SAGE code for posets and simplicial complexes. This will improve research in subgroup complexes, their nerves, and their topological properties as studied by UMN colleague Peter Webb. (Grinberg, Musiker, Webb)

Year 2: Implement KLR algebras and their modules, making connections with crystals by implementing Kashiwara operators on the modules. (Lee, Schilling, Scrimshaw)

Year 3: Implement KR modules [58, 59] and their crystal bases using the implementation of KR crystals [163, 132, 137] currently in SAGE. The (t-analogs of) q-characters of KR modules, which are a solution to T-systems, will also be added by using Scrimshaw's implementation of Frenkel–Mukhin [83, 84] and Nakajima [157, 159, 158] algorithms. (Grinberg, Lee, Musiker, Scrimshaw)

Year 3: Implement quantum groups for affine Lie algebras. (Brubaker, Lee, Scrimshaw)

Year 3: Implement root multiplicities of infinite dimensional Kac–Moody Lie algebras, weight multiplicities of their representations and Weyl-Kac characters of the representations. (Lee)

Years 3–4: In relation to this goal, create SAGE interfaces for REPS and CATREPS, which are packages for group representations in positive characteristic and representations of categories that are implemented in GAP by Webb, see [234]. (Grinberg, Musiker)

Years 4–5: Implement or expose constructions for homological algebra to expand on the code in SAGE that is essential to the representation theory of algebras at a more basic level. This will include constructing an interface for HOMALG [20], a package for homological algebra constructions that is implemented in GAP, and improving the interface to SINGULAR. (Musiker, Scrimshaw)

2.3. **Optimization, polyhedral geometry, and lattice point enumeration.** Optimization is a key technology in applied mathematics, the sciences, and engineering. We propose to develop new features that will make SAGE the go-to system for research, exposition, and teaching in discrete and mixed-integer optimization. In the optimization world, there is a well-established infrastructure of high-performance solvers (such as GUROBI, CPLEX), modeling systems/languages (such as AMPL [79]), and high-performance languages (recently, JULIA [39]). We will not attempt to duplicate or compete with these; SAGE already has interfaces to several of these systems.

Rather the focus will lie on topics with a high potential of synergy with other areas of mathematics. This potential is currently not realized in any existing interactive system; high-level mathematical systems such as MATHEMATICA or MAPLE do not have a strong following in optimization. Examples of such topics are primal methods in integer optimization (Gröbner bases of toric ideals [66, chapters 10–11], Graver bases [66, chapters 3–4]), geometry of numbers methods [66, chapter 2], nullstellensatz and positivstellensatz certificates [66, chapters 12–13], total unimodularity [230, 233, 232], lattice point counting [117], [66, chapters 5–9], cutting plane theory (polyhedral combinatorics) [62], and the theory of cut-generating functions [30, 31, 61, 25, 24, 63, 21, 22, 23].

As another aspect of this synergy, optimization methods (and the closely related polyhedral computation tools) are important in several areas of mathematics, including combinatorics, group theory, and commutative algebra. For example, the decomposition of certain representations of Lie algebras into irreducible components is given by integral points in a polytope [113, 164]. The duality theory of optimization provides various kinds of certificates, e.g., for verifying inequalities. Recent

advances in exact (certified) optimization solvers [64, 215, 65] have resulted in greater potential for optimization methods to give rigorous, rather than just numerical, results. There is also untapped potential in the combination with classic real-algebraic methods (QEPCAD B, section 5.2).

Köppe together with his Ph.D. students and with additional assistance by Scrimshaw plans to:
Year 1: Develop a Cython interface to Normaliz [50], which provides a superior implementation of polyhedral methods and lattice point enumeration.
Year 1: Provide a fast implementation of quasiperiodic piecewise linear functions and the spaces and algebras generated by these. This is the basis for later work on lattice-point-counting functions [17, 19], as well as on cut-generating functions [18], which both depend on this.
Years 2–3: Integrate and extend the electronic compendium of piecewise linear cut-generating functions [118] into Sage. Extend it to other models, such as the k-row Gomory–Johnson model [27, 29], the Yildiz–Cornuéjols model [239], and dual feasible functions [6].
Year 3: Develop a textbook view on the numerical solvers that hides implementation details such as the upper-bound method and provides textbook-style duality certificates.
Year 4: Develop an interface to state-of-the-art exact (rational) optimization software such as SoPlex [215] or QSopt-exact [65], as well as GLPK [144], and develop rational-reconstruction techniques, to provide rigorous duality certificates.
Years 4–5: Implement polyhedra given by parameterized inequalities [121]. Make lattice point counting functions for polyhedra depending on several real parameters [18] available. Replace existing proof-of-concept Maple code in LattE integrale [16] by efficient Sage code.

3. Lowering barriers for entry and developing educational tools

In order for Sage to reach its full potential, it is mission-critical for its base of regular users to broaden beyond the active researcher-programmers who contributed the bulk of its early development. We envision Sage as becoming a **standard open software tool**, used by undergraduates, graduate students, and professionals in increasingly sophisticated ways as they gain mathematical experience. As the user base increases, so will those interested in contributing both to the code and the many innovative educational applications that can be built on top of such a robust computational engine.

3.1. **Leveraging Sage for education.** The advent of SageMathCloud, which allows anyone to create an account and run Sage commands on a remote server, has made the power of Sage accessible as a tool for teaching. With just a browser, one can now access Sage and work on collaborative projects rather than having to go through the hurdle of installing Sage locally. An important aim of this grant is to build on the existing Sage **thematic tutorials** to develop further materials for use in undergraduate and graduate classes — even with advanced high-school students. The existing thematic tutorials are great introductions to the functionality of Sage on certain topics, such as symmetric functions and root systems, and we intend to build on this success.

Although the web has allowed easy sharing of free and open-license textbooks for decades now, most of these are still available largely as traditional pdfs or mostly-static HTML webpages, which fail to take advantage of **online interactivity**. We plan to use cutting-edge authoring tools, particularly MathBook XML [35] and XIMERA [60]. The former is a hybrid markup language, easily learned by anyone familiar with LaTeX and HTML, that allows a single source file to be output in a variety of forms suitable for printing or online use. More importantly, material such as proofs of theorems can be embedded as expandable "knowls," which reveal or hide themselves as needed to the reader who clicks on them. Sage examples can be embedded as cells, via the SageCellServer [185], which the reader can try out immediately, modify and try again, or cut and paste the code into a Sage worksheet elsewhere. Similarly, exercises can be embedded to

give students real-time formative assessment on their learning; XIMERA is particularly adept at this, providing easy LaTeX authoring tools and computing statistics on student success. A large body of educational research (e.g., [3, 42, 57, 68, 82, 86, 94, 106, 122, 142, 148, 166, 169]) points to the importance of engagement and interactivity in improving student learning, so this kind of technologically-created back-and-forth is highly desirable.

The PIs will incorporate SAGE in their teaching throughout the project and **collaboratively** develop teaching tools for use in the broader community. As tools are created, they will be shared by those teaching similar courses for feedback and adaption to different sites, leading to greater robustness from the outset. These can later be refined and improved by us and others, as described in Section 3.2. SAGE provides a invaluable avenue for students to come to grips with complicated computations and to see the power of theorems and conjectures in action.

Below is a timeline for some of our goals; we expect to produce additional tangibles as well.
Years 1–2: Convert existing thematic tutorials to a format that is easily accessible from the cloud, as editable worksheets rather than static webpages. (Lee)
Years 1–2: Develop a thematic tutorial for high school students, "Mathematics behind the Rubik's Cube", based on a "Mentor Connection" program at UConn. (Lee and Roby)
Years 1–2: Write new introductory thematic tutorials on permutations, Catalan objects, and combinatorial objects arising in statistical mechanics. These tutorials will be accessible both as teaching tools in college classes as well as tools for outreach to high school students, as discussed in Section 4.4. (Brubaker, Dilks and Striker)
Years 1–2: Update the cluster algebras and quivers compendium [156] developed under the previous OCI grant to include the recent explosion of new functionality and updates from the research community. Convert it to a hyperlinked thematic tutorial, designed to easily be updated as new SAGE code is developed. (Musiker)
Years 2–3: Develop interactive SAGE worksheets for working with cluster algebras, based on prototypes used in lecture in Fall 2016 and examples and exercises from previous semesters. (Musiker)
Years 2–3: Expand Schilling's SAGE thematic tutorial [202] to accompany Stanley's undergraduate textbook [214]. (Roby and Schilling)
Years 1–3: Collaboratively develop teaching tools in SAGE for applied linear algebra courses. Although we can build on the fine work Beezer has done in creating a model interactive linear algebra text for math majors [34], here we develop the standard sophomore-level course for a general STEM audience. Some of us will develop SAGE-coded examples, and others will rework them for most effective presentation using online tools. (Brubaker, Grinberg, Roby)
Years 2–4: Develop teaching tools for an undergraduate graph theory course, which take advantage of SAGE's large library of graphs and methods. (Grinberg, Scrimshaw)
Years 2–3: Develop SAGE teaching tools for optimization classes. Extend SAGE's textbook-style simplex tableau code so that it can be used to teach variants of the simplex method, the cutting-plane method, and branch-and-bound. Develop a textbook-style interface to state-of-the-art optimization solvers. (Köppe)
Years 4–5: Collaboratively develop interactive modules for abstract algebra and for enumerative combinatorics classes for undergraduates. (Musiker, Roby, Striker)

3.2. **Curating and disseminating teaching tools.** During the first year of the grant, we will create a webpage (e.g., linked from [217]), where we will store the learning tools and make them easily searchable by subject matter and type of tool. Code and examples may be stored locally or as links pointing to an author's website or github repository, as appropriate. In addition to actively curated high-quality examples (both our own and others), we will advertise this repository through the **sage-edu** mailing list, through meetings of the AMS and MAA, and through the NEXT fellow program aimed at early-career college professors.

4. Engaging and broadening the larger Sage community

The ten of us represent an active portion of the Sage developer community in algebra and combinatorics. We also plan to interface with the larger Sage developer community in other areas.

4.1. **Sage Days.** We are planning to hold five Sage Days meetings in the US associated to this grant, as well as two smaller Women in Sage Days (see Section 4.4) and one Sage Education Days. Sage Days are one-week workshops aimed to teach Sage to new users, to walk new developers through the development process, and, most of all, to collaboratively develop new features. Sage Days are extremely interactive, with much time devoted to design discussions and coding sprints. For each Sage Days we expect about 25–40 participants, ranging from undergraduate and graduate students to postdocs and senior researchers. The Sage Education Days will include participants who are active in pre-college education. The Sage Days in 2021-22 at UMN will include industry professionals and focus on their experience with Sage and mathematical functionalities of interest.

Year	Host	Theme (Organizers)
2017–18	UMN	Statistical mechanics (Brubaker, Grinberg, Musiker, Scrimshaw)
2017–18	UCD	Women in Sage Days (Schilling, Striker)
2018–19	NDSU	Lattice models & algebraic combinatorics (Dilks, Grinberg, Striker)
2019–20	UConn	Categorification & Lie (super-)algebras (Lee, Roby)
2020–21	UCD	Polyhedral geometry & optimization (Köppe, Scrimshaw)
2020–21	UConn	Sage Education Days (Lee, Roby)
2021–22	UMN	Connecting Mathematics and Industry (Brubaker, Musiker)
2021–22	NDSU	Women in Sage Days (Schilling, Striker)

As will be discussed in Section 7, Musiker, Schilling, and Scrimshaw have all had previous experience organizing such conferences. These previous Sage Days have resulted in many users (including Brubaker, Bryan Gillespie, Maria Gillespie, Striker and countless others) becoming developers, making their first contribution to Sage (and continuing to make contributions), fixing numerous bugs and related issues, adding many new features, and engaging in design decisions.

Each Sage Days has its own wiki page [197], which allows participants to add suggested features and report progress coming out of their coding projects both at and after the workshop. Summaries of the design discussions and decisions reached at each Sage Days are also included.

4.2. **Fostering Sage development through coding sprints.** In addition to the Sage Days, we are requesting funds in Years 2–5 for a program to maintain and strengthen our network of developers. We would provide opportunities for follow-up visits (as small group 3–5 days coding sprints) for participants of the Sage Days funded by this grant. We have secured space and support from both the Institute for Mathematics and its Applications (IMA) and the School of Mathematics at the University of Minnesota, where the Minnesota PIs will be available for technical and scientific support, but would require funds for the costs of participants that apply for follow-up visits. To encourage these coding sprints, priority projects would be identified by the community and applications solicited at said Sage Days.

As a prototype for what we have in mind, we note that the interface to the CHEVIE GAP package [89, 228], which implements complex reflection groups, was heavily discussed at Sage Days 64.5 and was one of eight priority topics identified at this meeting [192]. This code was subsequently finished during a focused 5-day 5-person coding sprint, i.e., Sage Days 80 [194].

Furthermore, we have backing from the larger Sage development community for such coding sprints. Brubaker and Musiker have co-authored a proposal to the IMA, located at the University of Minnesota, to sponsor and host 10–20 such Sage coding sprints open to developers and users in science and industry for 2017–18. If the IMA funds this proposal, applications would be selected

by a 9 member review board, which includes PIs Brubaker, Musiker, and Schilling, SAGE founder Stein, Henry Cohn from Microsoft Research and SAGE developers from other areas of mathematics such as arithmetic geometry and topology. The pre-proposal to the IMA met with enthusiastic reviews and the final proposal is pending approval. However, the IMA is only able to fund this program for 2017–18 with no possibility of renewal.

4.3. **RA support.** The undergraduate and graduate student RA support requested in this grant will expose beginning researchers to the powerful functionality available in SAGE. They will gain experience in basic coding to implement novel algorithms for research-level mathematics and computational science, leaving improved functionality in SAGE as a lasting byproduct. Moreover, our experience has shown that computing examples and developing an algorithm to solve a mathematical problem is one of the best ways to gain understanding of the intricacies of modern mathematics.

This project will also serve as a **gateway** to meeting collaborators on mathematical projects (including future Ph.D. advisors) and in related areas of mathematics and computational science. For example, Maria Gillespie met Schilling through SAGE Days, and the mathematical collaboration begun there is now funded through an NSF postdoctoral fellowship.

At all four institutions, both undergraduate and graduate students have been actively working with and contributing to SAGE over the last 3–5 years. These include PIs Dilks and Scrimshaw (now postdocs), along with many other students who have successfully moved on to positions in academia or industry.

4.4. **Increasing the participation of women in mathematics and computer science.** Mathematics and computer science are two subject areas that see the smallest percentages of women participants. SAGE has a great ability to demystify the coding process and make it more accessible to diverse populations. Striker has seen this first-hand. She became familiar with programming through attending SAGE Days, and as a result, gained the ability and the confidence to implement features into SAGE that were relevant to her research and to mentor others in SAGE programming. For the past two summers, Striker spent two weeks teaching ten North Dakota high school students, a majority of whom were women, about computational mathematics. She introduced them to programming and SAGE; many students said it was their favorite part of the entire summer program. The students began the program with no coding experience and completed it knowing that mathematics and computer science are accessible fields for them. SAGE has been the perfect platform for this learning opportunity. Striker will continue this summer outreach and, as a result of this grant, will write an introductory tutorial on this material; see Section 3.1.

The Women in SAGE workshops, organized by Schilling and Striker in Years 1 and 5, will build on the successes of previous such workshops by increasing the number of women in the community of SAGE developers. There have been Women in SAGE Days once per year since 2010; see [197]. Striker is an organizer of another such workshop in France in Jan. 2017 [195]. Women and people from underrepresented groups will be actively recruited to attend the other SAGE Days as well.

4.5. **International collaboration.** This project will work in **collaboration** with OpenDreamKit (see [161]), a Horizon 2020 European Research Infrastructure project funded for four years starting in 2015 (led by Nicolas Thiéry). The mission of OpenDreamKit is to build a sustainable ecosystem of open source mathematical software, which includes support for SAGE. OpenDreamKit will streamline access, distribution, and the portability of SAGE to a wide range of platforms, including high performance computers and cloud services, and improve user interfaces such as those through SAGEMATHCLOUD. This SAGE infrastructure support will complement our proposed activities and make our code highly visible within a broad interdisciplinary framework. In addition, there is an **international community** with members in Australia, Austria, Canada, France, Germany, India, Israel, Italy, Japan, South Korea, and elsewhere, that will contribute to and benefit from this grant's activities through the SAGE Days and newly-created code.

5. Delivering best practices to the wider open source math software community

As an innovation in mathematical software, we propose to provide **deeper integration** by spreading Sage's best practices for software development and community interaction to the wider open source math software community; in particular, for specialized scientific software.

5.1. **Issues and goals.** We plan to address the following specific issues and goals. Solutions to these problems are offered in Sections 5.2 and 5.3.

Sustainability and long-term viability. While the open source software world is thriving, there are many individual open source projects, even some of critical infrastructure importance, that have a very narrow developer base. A widely-publicized example is OpenSSL, which suffered serious security flaws as a result of a narrow developer base, which was unable to ensure sustained development and review of the software. This phenomenon is even more true in the world of mathematical software, where many projects essentially have one lead developer who is faculty or an academic researcher, together with a few members of his or her research group. For example, LiDIA [40, 140, 141], a sophisticated C++ library for number theory, under development for over a decade, all but disappeared from the web when the lead developer lost interest. Merely developing a Sage interface to the software has little effect on its development community. Köppe has first-hand experience with this, as the maintainer of the computational software packages 4ti2 and LattE integrale. These systems have been interfaced to various larger systems, such as GAP, Macaulay2, Polymake, Sage; but this integration has been a one-way street, except for the occasional bug report received from the maintainers of these systems.

Code quality, documentation, discoverability. Many essential libraries were developed before today's widespread best practices on software quality assurance. Consider QEPCAD B, a system for real algebraic computation and quantifier elimination in the first-order theory of the real numbers. It is the strongest open source system that implements partial cylindrical algebraic decomposition (PCAD); only the proprietary Mathematica system has a better implementation. However, QEPCAD B is severely underdocumented and has no active developer community.

Genericity without code duplication and community fragmentation. Even with Sage bridging the gaps between mathematical software ecosystems, the goal of "building the car" can conflict with the **genericity** expected of a general-purpose mathematical system. As a case study, consider polyhedral computation, an important tool in combinatorics, optimization, and commutative algebra. Sage uses PPL, a state-of-the-art and industrial-strength library. Yet, this library (as well as all other available polyhedral computation libraries implemented in C or C++) is limited to computation over rational numbers. However, in general-purpose mathematical software such as Sage, there is demand for computations with polyhedra over more general ordered fields. As a basic example, platonic solids require polyhedra over real algebraic numbers. Moreover, not all combinatorial types of polyhedra appear in rational polyhedra. As a consequence, Sage contains, in addition to an interface to PPL, a generic implementation of polyhedral computations – an apparent contradiction to the principle "don't reinvent the wheel." Thus, a user switching from a rational to an irrational algebraic polyhedron will pay two *performance penalties* – a reasonable one for the more costly basic arithmetic operations; but then also a huge one for using a basic implementation of an algorithm instead of one written by a domain expert. It could be countered that, in the spirit of open source development, the implementation in Sage would be improved over time. However, this rarely happens because of **fragmentation of the developer community**. State-of-the-art implementations are fast in part because of years of research and algorithms engineering in response to benchmark studies and new challenges from applications [10, 9, 119]. Hence, experts will not typically work on the Sage implementation because they have no incentive: they do not want to be associated to code that does not compete with the leading implementations.

5.2. **Engineering Sage as an added value for the upstream library developers.** The key to engaging the developers and users of the upstream library with the Sage community is to add a suitable incentive. We plan to initiate an effort to add function-level documentation and testing of the upstream library. The idea is that Sage, and specifically Cython, will play the role of a unit testing framework for the upstream library. By providing this infrastructure, and initiating a project to supply the documentation and tests, the upstream developer and user community can be broadened and will have an incentive to engage with the Sage community. Schilling and Mike Hansen were successful with this approach for Buch's upstream library LRCalc [51]. We propose a refined approach, in which the documentation and testing code will become part of the upstream library (rather than the Sage library) and only depend on Cython, not on all of Sage. Köppe plans to focus this work on QEPCAD B, mentioned above, as a target system, in **Years 3–5**. He plans to involve undergraduate and graduate students in this project in a software seminar.

5.3. **Pushing Sage's genericity and categories to upstream libraries.** Consider the C and C++ based implementations lrslib, PPL, and Normaliz, all of which are leading implementations of polyhedral methods. Some of these systems already have a potential genericity by the use of C++ templates. We plan to make Sage's general ordered fields available using C++ classes based on the Pynac class 'numeric' or the Boost::Python template classes. More generally, these classes would be generated from Sage categories; the category of ordered fields would be but one example. Köppe and Scrimshaw plan to focus this work on Normaliz as a target system in **Years 1–2**. It is a state-of-the-art implementation of polyhedral methods, which by far outperforms other implementations and is under active development by a small group. A genericized version of it would provide fast generic polyhedral computations for Sage.

We plan to document the technical and community aspects of the work on Normaliz so that the lessons learned can then be applied by other Sage developers to additional systems of interest.

6. Broader Impacts

Many of our goals, projects and activities involve broader impact components. Here, we organize those impacts into two broad categories and point to the relevant sections in the proposal.

Sharing and Dissemination. Enhanced infrastructure for research §2; public availability and dissemination of code §1.5; outreach to community and sharing/developing code through workshops §4.1; fostering engagement of broader community §4.2; sharing best practices with the broader developer community §5; international collaboration §4.5.

Human Resources and Training. Improved STEM education §3.1, §4.3 and educator development §3.2; globally competitive STEM workforce and increased partnership between academia and industry §1.4; increased participation of women in mathematics and computer science §4.4; training of community and students in the code §4.1; training of broader developer community §5.

Metrics. Sage development is done via Git branches submitted to the Sage trac server [227]. We will keep track of:

(1) trac tickets developed under this project using the tag CROSSPRODUCT,
(2) publications citing Sage at `http://www.sagemath.org/library-publications.html`,
(3) number of participants, new developers, new users, trac tickets from each Sage Days.

7. Results of Prior NSF Support

The PIs have been supported by the following NSF grants in recent years:

• OCI-1147161/OCI-1147247: Collaborative Research: Sage-combinat: Developing and Sharing Open Source Software for Algebraic Combinatorics (2012–2016) (Musiker, Schilling w/Bump, Stein)
• DMS-1401208: Conference on Representation Theory and Related Topics (2014–2015) (Lee)

• DMS-1406238: Metaplectic automorphic forms and matrix coefficients (2014–2017) (Brubaker)
• DMS-1320051: Infinite-Dimensional Relaxations of Mixed-Integer Optimization Problems (2013–2017) (Köppe)

In what follows, we will focus mainly on the Collaborative Research grant that was focused on Sage development, and on aspects of the above DMS grants most relevant to the proposal at hand. In addition, numerous other NSF grants have supported the general development of Sage over the past 10 years, see [184]. In particular, the Research Training Grant DMS-1148634 in Combinatorics at University of Minnesota (2012–2017) (co-PI Musiker with Pylyavskyy, Reiner, and Stanton) has supported Scrimshaw's postdoctoral fellowship and travel. This has lead to numerous developments and improvements to Sage, which aided in developing the results of [162, 167, 199, 201, 208].

7.1. **Intellectual merit.** The collaborative grant OCI-1147161/OCI-1147247 focused on Sage-Combinat, a subproject of Sage whose mission is "to improve Sage as an extensible toolbox for computer exploration in (algebraic) combinatorics and foster code sharing between researchers in this area." Many **new features** were implemented as part of this project including rigged configurations, new models for crystal bases, puzzles (used to count Littlewood–Richardson coefficients), various tableaux that arise in the combinatorics of the affine Grassmannian, branching rules for Lie algebras, the category framework, computations for cluster algebras, and new constructors for posets and signed permutations.

The **symbiotic relationship** between mathematical research and the code has been extremely powerful. As a characteristic example of the synergistic activities involved, we highlight a specific development (cf. [229]) which implements nonsymmetric Macdonald polynomials. As far as we are aware, there is no other mathematical software system which currently computes nonsymmetric Macdonald polynomials in this generality. This ticket was a truly collaborative effort carried out by many people at the ICERM program "Automorphic Forms, Combinatorial Representation Theory and Multiple Dirichlet Series" [172] in the Spring of 2013. This was immediately used in the paper [46] to give a representation theoretic interpretation of specializations of these polynomials.

In addition to these research-relevant functionalities, as part of this grant, we have co-organized and/or supported 8 **Sage Days**. Most of the Sage Days had about 25–40 participants (depending on the topic), with the Sage Days in Paris having over 50 participants (cf. [187, 196, 188, 189, 190, 191, 192, 193]). The workshops consisted of mathematical presentations, presentations on Sage and coding sprints. The mathematical presentations included tutorials on the relevant mathematics for the entire audience and more advanced talks for interested participants.

Mathematically, some of the features implemented at these workshops included (a) checking whether a lattice is atomic, semi-modular, or modular; (b) Littelmann path models for crystals; (c) quasisymmetric functions and nonsymmetric Macdonald polynomials; (d) construction of posets of semi-standard Young tableaux; (e) computations of characteristic and zeta polynomials of posets; (f) re-implementing cluster seeds more efficiently via F-polynomials rather than rational functions; (g) allowing other variable labels and other types of mutations; (h) presenting upper cluster algebras; (i) constructions of subword complexes; (j) basic implementation of fully packed loops; (k) improved support of tableaux classes; and (l) further extensions to alternating sign matrices.

In DMS-1320051, a classic infinite-dimensional relaxation of integer optimization problems by Gomory-Johnson has been made algorithmic. A comprehensive survey on this relaxation has been compiled to facilitate further research. The survey is integrated with interactive software [98] written in Sage, including an electronic compendium of extreme functions in the literature. The grant DMS-1406238 has produced new connections between lattice models and matrix coefficients for p-adic algebraic groups. Many of these findings were aided by Sage with explicit acknowledgment and code produced for these projects could be integrated into future Sage releases.

7.2. **Broader Impacts.** The Sage Days included a strong outreach component and have been a potent tool for connecting researchers and recruiting Sage users and developers. A number of participants of these Sage Days workshops have become more active Sage developers; of particular note are Emily Gunawan, Frederico Castillo, Maria Gillespie, and PIs on this current grant Brubaker, Dilks, Grinberg, Roby, Scrimshaw, and Striker.

Here is feedback given during some of the accompanying post-workshop surveys: **"**I learned how to use Sage and the whole editing process. This was like learning to fish rather than being given a fish.**"** **"**[I was] able to interact with many more experienced people who were able to teach me a lot of what I needed to know to work on my research goals.**"** **"**I was able to speak with Sage experts with whom I have mathematical interests in common. I hope what I learned will enable me to use Sage productively in my own research.**"** **"**The workshop was very inclusive. I was happy to meet people in my field and very happy to meet people at my level using Sage for similar tasks as I need.**"** **"**The idea of taking my proprietary code and contributing it to the open source community will be something I will work on that I would not have done beforehand.**"**

In addition to disseminating software features and actively engaging the research community at Sage Days, a considerable effort was made to make the code accessible by documenting and explaining it in **books** and **tutorials**. Lam, Lapointe, Morse, Schilling, Shimozono and Zabrocki wrote a book on "k-Schur functions and affine Schubert calculus" [126], which includes many examples in Sage. This book is freely available on the arXiv and the Springer website. In the summer of 2012, Schilling and Zabrocki did a major overhaul of the symmetric function code in a 15,000 line patch that also includes a new tutorial on symmetric functions.

Under the grant OCI-1147161/OCI-1147247, a number of graduate student RA's have also been supported. This model has been quite successful, both providing the graduate students with a deeper understanding of the mathematical material as they consider how to implement cutting edge algorithms or definitions, and leading to computations that are useful in their research. At UC Davis, Scrimshaw (now a PI on this grant and an extremely prolific Sage contributor) and Roger Tian were funded as graduate students. At the University of Minnesota, Alex Csar, Theo Douvropoulos, Dilks (now also a PI), Emily Gunawan, and Ben Strasser were funded. Dilks was a reviewer for several tickets and also started running the Patchbot system, which automatically attempts to apply tickets in progress and runs a full doctest suite. In DMS-1320051 and DMS-1406238, project funds have assisted 5 Ph.D. students and 4 undergraduates.

7.3. **Outcomes.** Here we give some further **highlights** regarding the outcome of grant OCI-1147161/OCI-1147247 (see also the annual reports on this grant for more details):

- A total of 314 patches were merged into Sage; many more are currently being developed [227].
- 34 papers/preprints/REU Reports [104, 162, 134, 205, 180, 179, 133, 152, 14, 204, 153, 203, 137, 13, 151, 136, 11, 12, 132, 135, 198, 199, 200, 207, 124, 138, 101, 92, 4, 5, 2, 145, 139, 100] and 2 books [126, 52] featuring Sage were written by Musiker, Schilling, their collaborators and students.
- Currently, at least 55 papers, 3 books, 4 theses, and 20 preprints cite or acknowledge the SageCombinat project. For more details and updated information see [186].

In DMS-1320051, 5 refereed journal articles [26, 28, 30, 31, 118], 3 book chapters [27, 121, 97], 2 submitted manuscripts [29, 119], 1 arXiv preprint [120], and 61 merged Sage tickets were written by Köppe and his collaborators and students. The open source software [98] with an electronic compendium of extreme functions was published on github [116]. DMS-1406238 has thus far resulted in 5 refereed journal articles [45, 47, 48, 46, 91] and 2 arXiv preprints submitted for publication.