The Clados Calculator

[and some ideas for improvement]

version 0.85

Interworld Transport Project
Dr Alfred W Differ
adiffer@gmail.com

Table of Contents

What the Calculator is intended to be

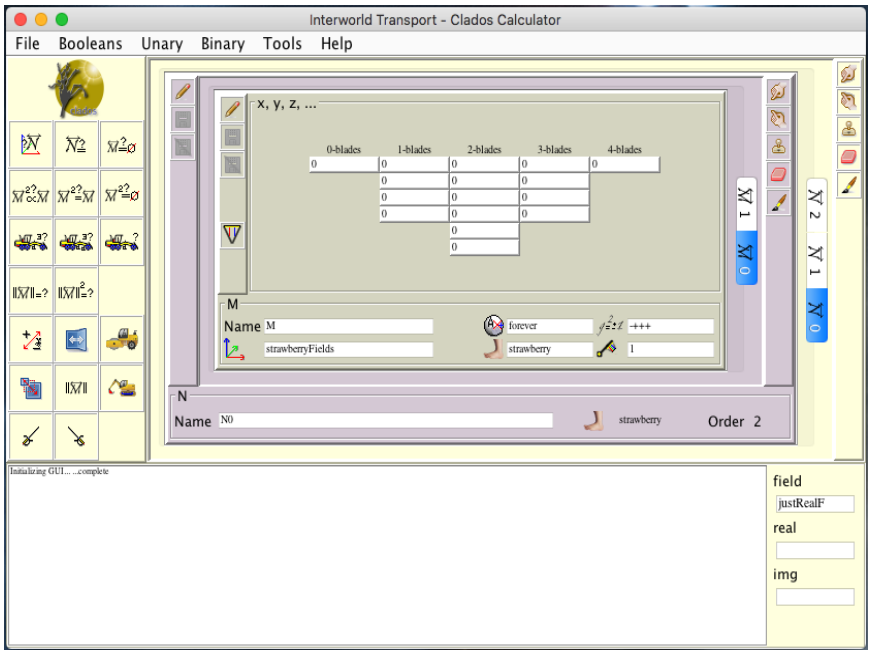
The Clados Calculator started as a way to exercise the functionality of the clados library, so its initial layout can be thought of as a collection of unit tests. Along the way, though, it also became the tool that facilitated feature expansion in clados. The addition of multi-algebra objects (multi-monads or multi-multi-vectors as nyads) showed up between version 0.5 and the present version. Some of Nyad's features were unclear in terms of requirements, so clados and Clados Calculator are progressing in parallel.

The Calculator is essentially a four-function calculator for geometric objects as expressed by clados. The geometry is expressed as Clifford algebras with a twist. The number 'field' used to represent geometric magnitude has some of the same structure as the geometry, so a single algebra using complex numbers can be written as an order two nyad. Because of this, the soft typing scheme used for monads in cladosG is repeated for numbers in cladosF and that impacts how numeric operations work in the calculator.

Hard object types are largely respected by the calculator. Very little casting occurs and when it does it involves conversion between java primitives in support of geometry constructors. The current calculator supports only real numbers expressed as java floats (single precision), so a complex number monad would have to be constructed as a nyad. Future versions of the calculator, however, will exercise all cladosF numeric classes.

Soft object types (tags) are in use to represent reference frames at run-time when it is far too late to create new static types. For example, two monads referencing tangent points for their coordinate basis sets both use Foot references, but must be able to distinguish between the feet to protect against improper comparisons and algebraic operations. Having a Foot is not enough. Which foot is referenced by a monad matters. Distinct objects are enough to ensure comparisons and operations are valid, but the strings contained with each Foot ensures the human reader can identify them. It's like naming points on a plane. They are all points, but might be distinct points.

A pure four-function calculator has arithmetic operations acting on pure magnitudes. CladosF is the package intended for division algebras typically used as magnitudes. Real and Complex numbers can be imitated with zero and one generator geometric algebras, but there isn't much point if the user does not intend to assign geometric meaning to generators. The situation is less clear with quaternions because two generators are required and they must anti-commute. If one sticks to division algebras for magnitudes and avoids giving meaning to any generators, the usual four functions apply. These four functions mean more in the calculator, though, because the objects being operated upon typically have geometric meaning. Only in the small division algebras do the meanings we learned in elementary school apply.



What the Calculator will NOT be

The calculator intentionally does not attempt to perform a number of functions that are already better handled elsewhere.

- Graphical Representation
- *Scripted Calculations*
- Type∫etting ╒unctions

Graphical representations of geometry described by multivectors are relatively straightforward for algebras with a small number of generators. However, the development work quickly escalates for higher algebras and mixed signatures. Development of such a feature would turn the application from a calculator to a presenter because the bulk of the code would be for drawing the geometry. Others may tackle such an effort if they wish for higher algebras, but we won't here. There are already good tools available for the simpler cases.

Scripted calculations involve development of a suitable programming or macro language. Since part of the purpose of the calculator is to act as a test platform for the Clados library, there is a conflict of purpose here. Besides, there are already suitable scripting environments in other tools, so it would make more sense to enable Clados to be plugins in those tools and re-use their well-developed and tested platforms.

Well developed presentations and print-ready layouts of objects and results for use in papers and publications involves suitable integration with adapters for external tools already in use by the community. This author has little interest in such things, but is open to any work others might contribute in these directions. One exception to this is the author's intent to develop a simple save and load routine so current objects in the stack may be stored and reloaded at a later time. The XML and JSON methods in the code today are a partial step in this direction.

THE BOOLEAN TESTS

Involving Monads

Button	Description
isReferenceMatch	This test takes the currently selected monad and tests it against the next one the stack below it. It should fail for monads in a nyad, but for nyads with only one monad in them, the next monad in the stack is in the next nyad. In that case the test might pass or fail. Reference frame matches test that Algebra references point to the same actual object. Algebras with the same name that are not the same actual object in memory will not pass. If algebra references match, frame name strings are compared next. All that is necessary at this time is that the strings be the same. If they are, the numeric objects are compared. If they refer to the same DivFieldType in memory, this test will pass. As with algebras, it is not enough that their names have the same string representation.
isZero	This test takes the currently selected monad and tests whether the magnitude of each blade is zero in the sense the magnitude objects understand the term. The calculator relies upon a monad static method to determine the result. Each magnitude 'coefficient' is responsible for determining whether it is an additive zero, so this static function largely keeps tally of the results to be combined.
isNilpotent	This test takes the currently selected monad and multiplies it by a copy of itself and then tests whether the result isZero. The calculator relies upon a monad static method to determine the result.
isIdempotent	This test takes the currently selected monad and multiplies it by a copy of itself and then tests whether the result isGEqual. The calculator internally upon a monad static method to determine the result. This method first detects whether the monad isZero before building the copy and doing the multiplication. The isGEqual method is not direction exposed to the calculator, but it tests for a reference match (which must pass due to the way the product is constructed) and then checks the resulting blade magnitudes one at a time. [This checking of every blade should be altered a bit when the sparse flag is set. In that case checking should be done for blades in grades known to be in the monad. TODO]
isIdempotent Multiple	This test takes the currently selected monad and multiplies it by a copy of itself and then tests whether the result isGEqual to within a scalar proportion. The calculator relies upon a monad static method to determine the result. This method first detects the monad isZero before building the copy and doing the multiplication. Once the product is formed it is tested to see if it isIdempotent before re-scaling the product using the inverse of the first non-zero magnitude of a blade to see if that version isGEqual to the monad being tested. [The current method for finding the first non-zero magnitude is a little lame. The monad's grade key should be used instead of the search method currently encoded. TODO]
isGrade	This test takes the currently selected monad and the real number in the input queue and answers the following question. Is the monad purely of that grade? Integer entries are required for the operation to make sense. A test against a zero will determine whether the monad is a pure scalar. The calculator displays a monad's grade key, so this function isn't visually necessary. However, the object's method must be able to answer the question without human inspection.
isMultiGrade	This test takes the currently selected monad and and answers the following question. Is the monad purely of any grade? Nothing in the input queue is necessary for this function to work. The calculator displays a monad's grade key, so this function isn't visually necessary. However, the

Button	Description
	object's method must be able to answer the question without human inspection.
hasGrade	This test takes the currently selected monad and the real number in the input queue and answers the following question. Does the monad contain that grade? Integer entries are required for the operation to make sense. A test against a zero will determine whether the monad has a scalar part. The calculator displays a monad's grade key, so this function isn't visually necessary. However, the object's method must be able to answer the question without human inspection.

Involving Nyads

Button	Description
IsReferenceMatch [Strong/Weak]	This test takes the currently selected nyad and tests it against the next one the stack below it. It should fail if there are any monads in either list that cannot pass a reference check with a monad in the other list. The strong form of this test requires the existing monad lists pair up exactly. The weak form of this test requires that the first nyad pair up exactly leaving open the possibility that the second nyad has unmatched monads. Nyad reference matching can be configured to require a match test to fail on unpaired monads that appear to share algebras because algebra names match yet the monads fail a reference match test. This switch is used to prevent nyad order increases when operations are used that should cause changes instead of concatenation.
isMEqual	This test takes the currently selected nyad and compares it against the next one in the stack. A few quick exit tests are tried like comparing the order of each nyad and the foot they should share if they are possibly equal, then the static method works down the monad list of the first looking for algebra match pairs and if one is found for isGEqual to pass. If every monad has a pair that passes, the method tests for reflexivity to complete the test. Equality is reflexive.
hasAlgebra	This test takes the currently selected monad and compares it to monads in the next nyad on the stack. If one of the monads in the next nyad uses exactly the same algebra reference this operation returns TRUE. Otherwise it returns FALSE. This operation in the calculator helps users deal with the fact that the calculator does not expose algebra references but does expose algebra names. If two algebras share a name, they need not be the same algebra when it comes to the objects involved. [It is best practice to avoid re-use of algebra names anywhere in a model, but clados will do nothing to encourage the good behavior.]
isScalarAt	[TODO This function isn't in the calculator yet, but should be added.] This test takes the currently selected nyad and the algebra name in the input queue and answers the following question. Is the nyad a pure grade scalar at the named algebra? String entries are required for the operation to make sense. If the nyad has a monad with an algebra by that name, that monad is tested to determine if it is a pure grade scalar with isGrade(0). The calculator displays a monad's algebra name and grade key, so this function isn't visually necessary. However, the object's method must be able to answer the question without human inspection.
isPScalarAt	[TODO This function isn't in the calculator yet, but should be added.] This test takes the currently selected nyad and the algebra name in the input queue and answers the following question. Is the nyad a pure grade pseudoscalar at the named algebra? String entries are

Button	Description
	required for the operation to make sense. If the nyad has a monad with an algebra by that name, that monad is tested to determine if it is a pure grade pseudoscalar with isGrade(max_grade). The calculator displays a monad's algebra name and grade key, so this function isn't visually necessary. However, the object's method must be able to answer the question without human inspection.

THE SIMPLE STATE TESTS

Involving Monads

Button	Description
whatGrade	This test takes the currently selected monad and outputs the result of the following question to the real field in what is normally the input queue. If the monad is of pure grade, what is it? Integer results are returned if the selected monad has a pure grade.

Involving Nyads

THE SIMPLE STATE MUTATORS

Involving Monads

Each monad is represented on a distinct tabbed pane within a panel representing a nyad. Each pane has on it set of buttons that represents operations specific to that particular monad. The buttons look the same on each panel, but they are not the same button objects in the calculator. They are {Edit, Save, Abort, Erase, Create}. All other monad specific mutators appear once at the top level of buttons on the calculator.

Button	Description
edit <|> **.edit.**	This mutator opens the fields for monad name, frame name, and those that represent magnitudes for the selected monad so they can be edited by the user. It also toggles the behavior of the edit button (shown when the button label changes to .edit. so the button will turn off the edit mode the next time. [Foot, signature, grade key, and algebra are not editable.] Entering new numbers for editable magnitudes does not adjust the underlying monad immediately. It is possible to alter the magnitudes visible on the pane used by the monad without them being pushed to the monad. The edit button ONLY opens the option to edit the fields.
save	This mutator saves the fields that represent magnitudes to the selected monad. The entire array of magnitudes is assumed to have been altered, thus they are all pushed into the monad. Also saved are the contents of the monad name and frame name. If the monad panel was in edit mode, it also ends edit mode.
abort	This mutator aborts any changes to the fields that represent magnitudes by overwriting them with the magnitudes in the currently selected monad. The entire array of magnitudes is overwritten by information in the monad. Also overwritten are the contents of the monad name and frame name. If the monad panel was in edit mode, it also ends edit mode.
copy	This mutator provides a way to create a new nyad that is a deep copy of the currently selected nyad. New monads are created, but references to algebras are re-used. References to DivFieldType's are re-used. A copied nyad should pass a strong reference match test and an equality test yet be editable without causing its source to change.
create	This mutator provides the way to append a new monad with zero magnitudes to the selected nyad. The calculator provides a means to copy a reference to an existing algebra used by another monad selected while the the create dialog is open. As long as an algebra chosen isn't already being used, the new monad will be appended to the original nyad's stack. If algebra re-use would occur when a monad is appended, an error results and no monad is appended. If no algebra is chosen for re-use, a new one is created based on the string provided in the dialog.
erase	This operation removes the currently selected monad from the nyad containing it.
gradeCrop TODO	This action takes the currently selected monad and the real number from the input queue and trims all the magnitudes in the monad to zero except for those associated with the grade represented by the real number. Integer entries for the real number are required for the operation to make sense. [Currently called gradePart] [[The calculator displays the magnitudes and users can edit them, so this function isn't strictly necessary. However, the object's method must be able to produce the result without human intervention.]]

Button	Description
gradeCut TODO	This action is the inverted version of gradeCrop. This one trims the magnitudes to zero if they are associated with the grade represented by the real number. Integer entries are required for the same reason as gradeCrop. [Currently called gradeSuppress]

Involving Nyads

Each nyad is represented on a distinct tabbed pane within a panel representing the nyad list. Each pane has on it set of buttons that represents operations specific to that particular nyad. The buttons look the same on each panel, but they are not the same button objects in the calculator. They are {Edit, Save, Abort, Erase, Create}. All other nyad specific mutators appear once at the top level of buttons on the calculator.

Button	Description
edit <|> **.edit.**	This mutator opens for edit purposes the nyad name. Foot and order are not editable. It also toggles the behavior of the edit button (shown when the button label changes to .edit. so the button will turn off the edit mode the next time.
save	This mutator saves the currently displayed nyad name overwriting its previous name. If the nyad panel was in edit mode, it also ends edit mode.
abort	This mutator aborts the currently displayed nyad name by overwriting it with the nyad's current name. If the nyad panel was in edit mode, it also ends edit mode.
copy	This mutator provides a way to create a new nyad that is a deep copy of the currently selected nyad. New monads are created, but references to algebras are re-used. References to DivFieldType's are re-used. A copied nyad should pass a strong reference match test and an equality test yet be editable without causing its source to change.
create	This mutator provides the way to append a new nyad with zero magnitudes for the single monad in its stack. The calculator provides a means to copy a reference to an existing foot used by another nyad selected while the the create dialog is open. If no foot is chosen for re-use, a new one is created based on the string provided in the dialog.
erase	This operation removes the currently selected nyad from the stack.

THE FOUR FUNCTIONS

Addition

In any algebra, addition involves a pair of objects that share the same reference frame and produces another object using the same reference frame. Technically speaking, addition is a map from a pair of objects to one object, but the calculator carefully restricts the map by enforcing reference frame matches on the operands. Addition of objects not in the same algebra is treated as having no meaning.

Addition for monads follows the same rules as the addition operator defined in a Clifford algebra except for the additional reference frame checks. If the monads do not share the same algebra object, they fail the reference check. In practice, this is handled with a static method associated with the correct monad class. This method compares algebra references in the pair to ensure they are the same. At present, the method also compares frame names which act as a soft tag alias for an algebra. Finally, the method also checks the soft tag associated with the magnitudes represented by the DivFieldType object. If the magnitudes reference the same DivFieldType, they pass this piece of the reference frame match test. Be forewarned, though. What gets checked within this static method may change in the future as a physics library counterpart to clados is written. The purpose of the reference match test is to prevent non-physical operations from happening.

If the reference frame test passes, the addition operation numerically adds magnitudes for similar blades. Multigrade results are quite possible. Results that are idempotents or nilpotents are also possible. Numeric addition occurs by delegation when the monad's operation finds similar blades and calls the correct method for adding the magnitudes in use.

The final complexity arises in defining addition for nyads within clados. Since this class is not currently constrained by well known geometric expectations, it is currently defined on a per monad basis. A nyad is essentially a list of monads, so nyad addition is defined as a similar binary map involving lists where any monad pairings from the lists that can be found that pass a reference frame match test are added in the sense of monads. Clados currently requires monads not to pass a reference frame match test before they can be inserted into a nyad's list, so this ensures addition of nyads is not indeterminate. If a pairing exists between monads on each list, neither monad will pair with any other.

Finally, nyad addition does not require each nyad's monad have a corresponding pair. An assumption is made for unpaired monads where the nyad with the missing match actually has a match with a new zero scalar monad. That means nyad addition can produce a result with more monads in the list than either nyad had to start. In an extreme case with no pairs, the result will have a list that simply concatenates the lists of the two nyads. At the other extreme, the result will have exactly the same order as the nyad with the largest order because algebra pairs are found for all monads in one of the nyads. That means nyad addition as an operation ranges from list concatenation to something that looks like n-tuple addition.

Subtraction

In any algebra, subtraction involves an ordered pair of objects that share the same reference frame and produces another object using the same reference frame. Technically speaking, subtraction is a map from an ordered pair of objects to one object, but the calculator carefully restricts the map by enforcing reference frame matches on the operands as it does for addition.

Monad subtraction works as one would expect since it must function as an inverse function for addition. Internally, monads delegate magnitude subtractions to their cladosF elements the same way they do for addition. At no point is there a multiplication by "-1" and then an addition, though, unless it occurs deep within a JVM involving java primitives. Multiplication is a distinct operation with geometric meaning and not to be confused with addition.

Nyad subtraction does not require each nyad's monad have a corresponding pair for the same reason as with addition. The same assumption is made where a 'zero' is invented in one list to create the missing member of a pair. That means nyad subtraction is, as should be expected, an operation that ranges from list concatenation to n-tuple subtraction.

Multiplication

In any algebra, multiplication involves an ordered pair of objects that share a reference frame and produces another object using the same reference frame. Technically speaking, multiplication is a map from an ordered pair of objects to one object, but the calculator carefully restricts the map by enforcing reference frame matches on the operands as it does for addition.

Multiplication of monads follows the rules for multiplication defined in the objects used as magnitudes as well as the rules for multiplication in a Clifford algebra except for the additional reference frame checks. This is dictated by reasonable expectations for elements of a Clifford algebra. Reference frame checking is done the same way as for addition meaning the referenced algebras must match, frame name strings must be the same, and the magnitudes must share a reference to a DivFieldtype.

If reference frame tests pass, the multiplication operation numerically multiplies magnitudes and geometrically multiples blades. Multigrade results are quite possible. Idempotent and nilpotent results are too. Multiplication of multi-grade monads is distributive, so every blade in one monad is multiplied against every other blade in the other monad. Numeric multiplication occurs by delegation while blade multiplication is handled by an algebra's gproduct object. A monad's multiplication operation mostly keeps a tally of results collected from the others. The purpose of the reference frame test is most easily seen with multiplication because a single algebra object can agree with itself on how to multiply two pieces of geometry.

The final complexity arises in defining multiplication for nyads within clados. As with addition, multiplication is defined as a binary map involving lists where any monad pairings from the lists that can be found that pass a reference frame match test are multiplied in the sense of monads. For unmatched monads, an assumption is made where the nyad with the missing partner actually has a match with a unit scalar. That means nyad multiplication can produce a result with more monads in the list than either nyad had to start just like addition can. That means nyad multiplication is another operation that ranges from list concatenation to something that looks like n-tuple products involving determinants. The shared algebras keep track of all the details, though.

Division

In any algebra, division involves an ordered pair of objects that share a reference frame and produces another object using the same reference frame if the second member of the pair has an inverse. Technically speaking, division is a map from an ordered pair of objects to one object, but the calculator carefully restricts the map by enforcing reference frame matches on the operands along with a few other tests. Division also reveals how many have an overly simplistic understanding of both multiplication and division operations taught in elementary school and the calculator has to deal with this.

Division of monads follows the rules for inverse multiplication defined in the objects used as magnitudes and will attempt the rules for inverse multiplication in a Clifford algebra too with additional reference frame checks. Monads don't actually have methods for a division operation, though, so the complexity is being handled in the calculator. The second monad is checked a number of ways beyond the reference frame test including the list of boolean operations visible on the front of the calculator.

1. If the second monad is a zero scalar, it will have no inverse for a numeric reason. Zero has no multiplicative inverse. The division operation completes, however, by return the NaN value in the magnitude objects of the first monad.
2. If the second monad is nilpotent, it will have no inverse for geometric reasons. Nilpotents squares vanish, so they are a geometric equivalent for zero. The division operation completes, though, by returning the NaN value in the magnitude objects of the first monad.
3. If the second monad is idempotent, it will have no inverse for geometric reasons, but it does not behave like a zero scalar. Idempotent squares are the same idempotent, so they behave more like multiplication's identity element, yet they are not that element. To make matters worse, there is no element in an algebra to be multiplied against an idempotent that 'factors' it from the first monad, thus there is no path forward for the division operation to complete. Attempts within the calculator to divide by an idempotent will produce errors.
4. If the second monad is proportional to an idempotent, the proportion might have an inverse while the idempotent will not. The division operation will produce errors in this case because there is no path to completion.

The final complexity arises in defining division for nyads within clados. As with multiplication, it is defined as a binary map involving lists where any monad pairings from the lists that can be found that pass a reference frame match test are divided in the sense of monads. For unmatched monads, an assumption is made where the nyad with the missing partner actually has a matching unit scalar. That means nyad division can produce a result with more monads in the list than either nyad had to start just like multiplication can. That means nyad division is another operation that ranges from list concatenation to something that looks like n-tuple products involving determinants. The shared algebras keep track of all the details, though.

The seemingly special cases where division won't complete demonstrate that division isn't really about factoring a product. It is about measuring a product. A rectangle with sides A and B has an area of A*B. Dividing that area by a number C has an ambiguous meaning. Are we parting the rectangle into C pieces? Are we scaling side B by C and finding out A's magnitude by the new measure? The units that are assigned to the result of the operation would tell us because the first would have area dimensions and the second would have linear distance dimensions. When dividing by geometric elements, though, there is no ambiguity. The dimensions are part of the operation. Doing this brings with it the following limitations. Idempotents may not be used for scaling a measure. There are other things in an algebra that behave like zero. If care is taken with these possibilities, however, the division operation will delivery geometrically meaningful results.