$\mathcal{GA}$lgebra Primer

Alan Macdonald
Emeritus Professor of Mathematics
Luther College
Decorah, Iowa USA
mailto:macdonal@luther.edu
faculty.luther.edu/~macdonal

Abstract

This document describes the installation and basic use of the geometric algebra/calculus Python module *$\mathcal{GA}$lgebra* written by Alan Bromborsky. It was written to accompany my texts *Linear and Geometric Algebra* and *Vector and Geometric Calculus*.

This is only an introduction to the module; many features are not covered. In some situations there are simpler approaches to those described here. But to include them would complicate this introduction. For complete documentation see GAlgebra.pdf, which is distributed with $\mathcal{GA}$lgebra.

I encourage feedback and will post updated versions of this document as appropriate.

Printed
April 22, 2015

Contents

1 Installation

It is your responsibility to have $\mathcal{GA}$lgebra installed on your laptop before the summer school starts. No time will be spent in the computer sessions for this. If you are not able to install the software with the directions below, find a local person to help you.

Python is a programming language. SymPy is a computer algebra system written in Python. It provides *symbolic* computation capabilities. For example, it will solve $x^2 - 5x + 6 = 0$ for x. $\mathcal{GA}$lgebra adds symbolic geometric algebra and calculus capabilities to SymPy.

You will need Python, SymPy, Geany, and $\mathcal{GA}$lgebra. IPython and mpmath are optional. All are free, multiplatform, and downloadable from the web.

Python. Install the latest Python 2.7 version from `https://www.python.org/downloads/`. (Python 3 will not work.) When the "Customize Installation" window appears, choose to install pip. The Doc folder of a Python installation contains documentation. The online documentation is at `https://docs.python.org/2/download.html`.

If you already have Python but not pip (it is included with Python 2.7.9), there are instructions to install pip at `https://pip.pypa.io/en/latest/installing.html`.

SymPy. The program pip downloads and installs Python packages. To install SymPy, open a console and run **`pip install sympy`**.

SymPy capabilities: `https://en.wikipedia.org/wiki/SymPy`. Full documentation: `http://docs.sympy.org`. Tutorial, including IPython Notebooks: `https://asmeurer.github.io/scipy-2014-tutorial/html/index.html`

$\mathcal{GA}$lgebra. Go to `https://github.com/brombo/galgebra`. Download the ZIP file. Python has a subfolder Lib\site-packages. Copy the subfolder *galgebra* of galgebra-master to it.

There is a file setgapth.py in the galgebra folder. Open a terminal in that folder. On a Windows system run "python setgapath.py". On Linux and OS X systems enter and run the command "sudo python ./setgapth.py".

Barcelona testers: I have sent the conference organizers a newer version of $\mathcal{GA}$lgebra than that at github. Please use the newer version.

Geany. Geany is a program editor: type in a program, e.g., the one at the start of Section 2.1 and press F5 to run it. Download and install Geany from `http://www.geany.org/`. Choose the 32- or 64-bit version appropriate for your computer.

For printing to a console/terminal (see Section 4) Geany must know the location of the console and configure it. This happens automatically on OS X and Linux, but not Windows. For Windows download and install ConEmu (`http://conemu.github.io/`). In Geany go to Edit/Preferences/Tools/Terminal and enter the full path (your choice) of conemu's exe file (in quotes), followed by "/WndW 180 /cmd %c" (no quotes).

mpmath. Use mpmath for numerical (not symbolic) calculations. There are some algorithms available numerically in mpmath but not available symbolically in SymPy. One relevant to $\mathcal{GA}$lgebra is documented here.

Download: `http://mpmath.org/`. Documentation: `http://mpmath.org/`.

IPython notebook. The IPython notebook provides a way to do Python programming *interactively*: type a Python expression, press shift+enter, and the expression is quickly evaluated and displayed in the notebook with LaTeX formatting. IPython runs in a web browser.

To install IPython and the notebook run **`pip install "ipython[notebook]"`**. See Section 5 for information about the notebook.

2 Linear and Geometric Algebra

Notation. This document will use lower case italic for scalars (e.g., s), lower case bold for vectors (e.g., $\mathbf{v}$), upper case bold for blades (e.g., $\mathbf{B}$), and upper case italic for general multivectors (e.g., M). Python statements will appear in `this font`.

2.1 Linear Algebra

Type the Python program below into Geany. The program defines the matrix $M = \left[\begin{smallmatrix} 1 & m \\ 3 & 4 \end{smallmatrix}\right]$, prints it, and then computes and prints M^{-1}. The first line gives the program access to SymPy.

```
from sympy import *
m = symbols('m', real=True)  # Anything following a ''#'' is a comment
M = Matrix( [ [1,m],[3,4] ] )  # Extra spaces inserted for clarity
print M
print M.inv()
```

Press F5 in Geany to see the output:
[1, m]
[3, 4]
[1+3m/(4-3m), -m/(4-3m)]
[-3/(4-3m), 1/(4-3m)]
Thus

$$M^{-1} = \begin{bmatrix} 1 + \frac{3m}{4-3m} & \frac{-m}{4-3m} \\ \frac{-3}{4-3m} & \frac{1}{4-3m} \end{bmatrix} = \frac{1}{4-3m} \begin{bmatrix} 4 & -m \\ -3 & 1 \end{bmatrix}.$$

SymPy is a *symbolic* computer algebra system. We have used the symbol `m` in M. Symbols must be *declared.* One way to do this is with a `symbols` statement, as above. You can declare several symbols at once, e.g.,
`x1,x2,m,z = symbols('x1 x2 m z', real=True).`

Elementary matrix methods. SymPy provides several:

```
M + N      # sum
M * N      # product
M.inv()    # inverse
M.T        # transpose
M.det()    # determinant
M.rank()   # rank
```

Vector methods: norm, inner product.
One way is to implement vectors as matrices.

```
u = Matrix([1,2,3])  # A vector
v = Matrix([4,5,6])  # A vector
```

`print u.norm().evalf(3)`
  Output: 3.74
`print u.dot(v)`
  Output: 32

Span. The `rref` method computes a basis for the span of the row vectors of a matrix. ("`rref`" is an abbreviation for *reduced row echelon form.*)

```
A = Matrix([ [1,2,-1], [-2,1,1], [0,5,-1] ])
print A.rref()[0]
```

Output (condensed): ([1, 0, -3/5] [0, 1, -1/5] [0, 0, 0])

The vectors `[1, 0, -3/5]` and `[0, 1, -1/5]` form a basis for the span of the three row vectors of A.

Least squares.

```
x = Matrix([[ 0, 1], [ 1, 1], [ 2, 1], [3, 1]])
y = Matrix([[-1], [0.2], [0.9], [2.1]])
LS = x.solve_least_squares(y)
print N(LS, 3)   # 3 significant figurees
```

Output: [1.] [-0.95]   (Least squares line: $y = 1x - 0.95$)

```
print correlation(u,v)
```

Output: 1

Characteristic polynomials.

```
x = symbols('x')
M = Matrix([ [1,2], [2,1] ])
charpoly = (x*eye(2) - M).det() # eye(2) = 2 x 2 identity
print charpoly
```

Output: x**2 - 2*x - 3

```
print factor(charpoly)
```

Output: (x - 3)*(x + 1)

Singular value decomposition.

```
from mpmath import *
mp.dps = 4
A = matrix([[2, -2, -1], [3, 4, -2], [-2, -2, 0]])
```

`U, S, V = svd_r(A)`. _r for real matrix; _c for complex

Simplify trigonometric expressions. Use the function `trigsimp` (which is not perfect). For example,

```
x = symbols('x')
print trigsimp(sin(x)**2 + cos(x)**2)
```

Output: 1

SymPy "Helpers"

There are several SymPy routines in the $\mathcal{GA}$lgebra module. To use them, include this statement in your program: `from mv import *`

Systems of linear equations. `rref` (described above) also solves systems of linear equations. In this context the output from `rref` is not well formatted for human readers. The function `printrref` assumes that `rref`'s output is from a system of equations and prints it in a readable form.

As an example, consider the system $\left[\begin{smallmatrix} 1 & 2 & -1 & 2 \\ -2 & 1 & 1 & 0 \\ 2 & 0 & -2 & 4 \end{smallmatrix}\right]\left[\begin{smallmatrix} w \\ x \\ y \\ z \end{smallmatrix}\right] = \left[\begin{smallmatrix} 4 \\ -1 \\ 1 \end{smallmatrix}\right]$. The *augmented matrix* of the system consists of the coefficient matrix augmented with the column vector on the right side. Assign it to `A` and `printrref` it:

```
A = Matrix([ [1,2,-1,2,4], [-2,1,1,0,-1], [2,0,-2,4,1] ])
printrref(A, 'wxyz')
```

Output:
```
1w + 0x + 0y + -2z = 9/4
0w + 1x + 0y +  0z = 7/4
0w + 0x + 1y + -4z = 7/4
```

The output is another system of equations. This system has two important properties. First, it has the same solutions as the original. Second, the solutions can be read directly from its equations. Starting from the first equation of our example, $w = 2z + 9/4$, $x = 7/4$, $y = 4z + 7/4$, with z not further constrained. Set it equal to t. Then the solution is shown at the right.

$$\begin{bmatrix} w \\ x \\ y \\ z \end{bmatrix} = \begin{bmatrix} 2 \\ 0 \\ 4 \\ 1 \end{bmatrix} t + \begin{bmatrix} 9/4 \\ 7/4 \\ 7/4 \\ 0 \end{bmatrix}$$

Eigenvalues and eigenvectors. SymPy provides `M.eigenvects()` for the eigenvectors of matrix M. But its output is not well formatted for human reading. The statement `printeigen(M)` will print the eigenvalues of a matrix M, their multiplicities, and their eigenvectors.

Gram-Schmidt orthogonalization. It is applied to a list of vectors, each implemented as a matrix:

```
L = [Matrix([1,2]), Matrix([3,4])]
print GramSchmidt(L)
```

Output: [[1] [2], [4/5] [-2/5]]

A second argument set to `True` will normalize the eigenvectors:

```
print GramSchmidt(L, True)
```

Output: $[\,[\sqrt{5}/5]\,[2*\sqrt{5}/5], [2*\sqrt{5}/5]\,[-\sqrt{5}/5]\,]$

This output is not well formatted for human readers. `printGS` will print the output of GramSchmidt in decimal form:

`printGS(L, True)` (Note change from earlier version)

Output: [[0.447, 0.894] [0.894, -0.447]]

2.2 Geometric Algebra

To use the geometric algebra facilities of $\mathcal{GA}$lgebra, first create a specific geometric algebra. This code creates the standard 3D geometric algebra and names it g3:

```
from sympy import *
from ga import Ga  # import galgebra
g3coords = (x,y,z) = symbols('x y z')
     # Two sets of coordinate names: (x y z) in program, 'x y z' for printing.
g3 = Ga('ex ey ez', g=[1,1,1], coords=g3coords)
     # Create g3
     # 'ex ey ez': printing basis vector names
     # [1,1,1]: norms squared of basis vectors (assumed orthogonal)
(ex, ey, ez) = g3.mv()
     # (ex ey ez): program basis vector names
```

The two sets of coordinate names above, (x y z) and 'x y z', are the same. The same is true for basis vector names, (ex ey ez) and 'ex ey ez'. See Section 4 for reasons to make them different.

The lines above produce no output. Add these lines:

```
A = y*ex + 3*ex*ey
B = x*ey
print A*B
```

Output: 3*x*ex + x*y*ex∧ey.

Substitute. Sometimes you want to substitute specific values for variables. Example: `print (A*B).subs(x:1,y:2)` produces 3*ex + 2*ex∧ey.

Algebras.py. The file contains code to create many different geometric algebras, including g3 and all others used in this document, as well as the homogeneous, spacetime, and conformal algebras. Instead of typing the code for a geometric algebra into your program, copy it from this file.

Precedence. If you see the arithmetic expression $2+3*4$ you know to multiply $3*4$ first and then add 2. This is because mathematics has a *convention* that multiplication comes before addition; multiplication has higher *precedence* than addition. If you want to add first, write $(2+3)*4$.

The table shows the $\mathcal{GA}$lgebra arithmetic operators. They are given in precedence order (imposed by Python), high to low. Plus and minus are grouped because they have the same precedence, as do $<$ and $>$.

$*$	geometric product
$+\,-$	add, subtract
$\wedge$	outer product
$\vert$	inner product
$<\,>$	left, right contraction

The high precedence of $+-$ causes a problem. Consider the simple expression $\mathbf{u}+\mathbf{v}\cdot\mathbf{w}$. $\mathcal{GA}$lgebra evaluates it as $(\mathbf{u}+\mathbf{v})\cdot\mathbf{w}$. If you intend $\mathbf{u}+(\mathbf{v}\cdot\mathbf{w})$, as you probably do, then you must use the parentheses. As another example, $\mathcal{GA}$lgebra evaluates $\mathbf{u}\cdot\mathbf{vw}$ as $\mathbf{u}\cdot(\mathbf{vw})$. If you intend $(\mathbf{u}\cdot\mathbf{v})\mathbf{w}$, then you must use the parentheses. (For many authors $\mathbf{u}\cdot\mathbf{vw}$ does mean $(\mathbf{u}\cdot\mathbf{v})\mathbf{w}$.)

As a general rule, you must put parentheses around terms with inner or outer products, to "protect" them from the high precedence $\pm$'s bounding them, as in the $\mathbf{u}+\mathbf{v}\cdot\mathbf{w}$ example. And remember that within terms the geometric product has higher precedence than the inner and outer products, as in the $\mathbf{u}\cdot\mathbf{vw}$ example.

Multivector functions. In the table, `M` is a multivector; `A, B` are blades; `v1, v2` are vectors; and `MV` is a geometric algebra.

`MV.I()`	Outer product of basis vectors. Not necessarily normalized.
`dual(M)`	M^*. Returns MI. If you want MI^{-1} ****
`even(M)`	Even grades of M
`exp(M)`	e^M. M^2 must be a scalar constant. If $M^2 > 0$, use `exp(M,+)`
`grade(M,r)`	$\langle M \rangle_r$
`grade(M)`	$\langle M \rangle_0$
`inv(M)`	M^{-1}
`norm(M)`	$\lvert M \rvert$
`norm2(M)`	$\lvert M \rvert^2$
`odd(M)`	Odd grades of M
`proj(B,A)`	$\mathsf{P}_{\mathbf{B}}(\mathbf{A})$.
`rot(itheta,A)`	$\mathsf{R}_{\mathbf{i}\theta}(\mathbf{A})$.
`refl(B,A)`	$\mathsf{F}_{\mathbf{B}}(\mathbf{A})$.
`scalar(M)`	$\langle M \rangle_0$
`com(A,B)`	Commutator: $[A, B] = AB - BA$
`cross(v1,v2)`	Cross product
`Nga(M, prec=k)`	Round decimals in M to k significant figures.
`rev(M)`	$M^\dagger$
`ReciprocalFrame(basis)`	basis is a list of vectors enclosed in parentheses.
`MV.r_basis`	List of reciprocal basis vectors of MV basis. Each is expanded in MV basis.

There are also member function versions of the multivector functions, e.g., `M.even()`.

Linear transformations. The following two examples create a linear transformation (outermorphism) `L` on the geometric algebra **g2**. The matrix of the transformation with respect to the basis {ex, ey} is also shown.

`L = g2.lt('A')`. Matrix: $\begin{bmatrix} A_{xx} & A_{yx} \\ A_{xy} & A_{yy} \end{bmatrix}$ (because **g2** has coordinates x and y).

`L = g2.lt([[a,b],[c,d]])`. Matrix: $\left[\begin{smallmatrix} a & b \\ c & d \end{smallmatrix}\right]$.

An optional second parameter `f=True` makes the linear transformation a function of the coordinates.

If M is a multivector (not necessarily a vector), then $\mathsf{L}(M)$ is the result of the outermorphism `L` applied to M.

Linear transformations can be added (+), subtracted (−), and composed (*). `L.det()` (determinant), `L.adj()` (adjoint), `L.tr()` (trace), and `L.matrix()` are also available.

General Multivectors

$\mathcal{GA}$lgebra can create multivectors with *general* coefficients. For example, this code creates and prints a general vector Python variable V:

```
V = g2.mv('V', 'vector')
print V
```

Output: V__x*e__x + V__y*e__y

The double underscore __ is explained in Section 4.

An optional third parameter `f=True` makes the coefficients functions of the coordinates, i.e, makes the multivector a function of the coordinates. Here are the options for the first and second parameters (s is a string, n an integer):

First	Second	Result
s	'scalar'	scalar
s	'vector'	vector
s	'bivector'	bivector
s	n	grade n multivector
s	'pseudo'	pseudoscalar
s	'even'	even multivector (spinor)
s	'mv'	general multivector

The scalar result in the top row is a scalar multivector, a member of the geometric algebra. It is different from a SymPy scalar.

In addition, `g2.mv(c)`, where `c` is a scalar, is available.

General multivectors can be useful to test a conjecture about geometric algebra, especially when first learning. For example, let $\mathbf{B}$ be a bivector. After finding that $\mathbf{v} \cdot \mathbf{B}$ is in $\mathbf{B}$ for several vectors $\mathbf{v}$, one might wonder if this is always so. Assuming that we are using the standard 3D geometric algebra, the following code proves this.

```
v = g3.mv('v', 'vector')   # Construct a symbolic vector in g3.
B = g3.mv('B', 'bivector') # Construct a symbolic bivector in g3.
W = (v < B) ∧ B  # Test:  is zero  ⇔ v · B ∈ B.
print W.simplify()
```

Output: 0.

Thus in g3 at least, the vector $\mathbf{v} \cdot \mathbf{B}$ *is* in $\mathbf{B}$. This might encourage you accept provisionally that this is true in *all* dimensions. Or you might try to prove it. Try adding code to show that $\mathbf{v}$ is orthogonal to $\mathbf{v} \cdot \mathbf{B}$.

3 Vector and Geometric Calculus

3.1 Vector Calculus

Differentiation, including partial differentiation.

```
x,y = symbols('x y') # Define the symbols you want to use.
print diff(y*x**2, x)
```
Output: 2*x*y
```
print diff(diff(y*x**2,x),y)
```
Output: 2*x

Jacobian. Let X be an $m \times 1$ matrix of m variables. Let Y be an $n \times 1$ matrix of functions of the m variables. These define a function $\mathbf{f}\colon X \in \mathbb{R}^m \mapsto Y \in \mathbb{R}^n$. Then `Y.jacobian(X)` is the $n \times m$ matrix of $\mathbf{f}'_{\mathbf{x}}$, the differential of $\mathbf{f}$.

```
r, theta = symbols('r theta')
X = Matrix([r, theta])
Y = Matrix([r*cos(theta), r*sin(theta)])
print Y.jacobian(X)        # Print 2 × 2 Jacobian matrix.
print Y.jacobian(X).det()  # Print Jacobian determinant (only if m = n).
```

Sometimes you want to differentiate `Y` only with respect to some of the variables in `X`. Then replace `X` in `Y.jacobian(X)` with only those variables. For example, `print Y.jacobian([r])` produces the 2×1 matrix $\left[\begin{smallmatrix}\cos\theta \\ \sin\theta\end{smallmatrix}\right]$.

Integration. `integrate(f, x)` returns an indefinite integral $\int f\,dx$.
`integrate(f, (x, a, b))` returns the definite integral $\int_a^b f\,dx$.

```
x = Symbol('x')
print integrate(x**2 + x + 1, x)
```
Output: x**3/3 + x**2/2 + x

Iterated integrals.
This code evaluates $\int_{x=0}^{1}\int_{y=0}^{1-x}(x+y)\,dy\,dx$:

```
x, y = symbols('x y')
I1 = integrate(x + y, (y, 0, 1-x))
I2 = integrate(I1, (x, 0, 1))
```

evalf.

```
print log(10), log(10).evalf(3)
```
Output: log(10) 2.30

3.2 Geometric Calculus

Curvilinear coordinates. Curvilinear coordinates are implemented by creating an appropriate geometric algebra. For example, this code creates sp3, the standard geometric algebra in $\mathbb{R}^3$, in spherical coordinates:

```
sp3coords = (r, phi, theta) = symbols('r phi theta')
sp3 = Ga('er ephi etheta', g=None, coords=sp3coords, \
  X=[r, r*sin(phi)*cos(theta), r*sin(phi)*sin(theta), r*cos(phi)])
```

The "\" is Python's line continuation character.

Mathematics naming convention: ϕ colatitude, θ longitude.

```
(er, ephi, etheta) = sp3.mv()
```

Here is a different `Ga` statement to create sp3:

```
sp3 = Ga('er ephi etheta', g=[1,r**2,r**2*sin(theta)**2], sp3coords)
```

Gradient operator. `grad = g3.grad` assigns to `grad` (your choice) the gradient operator of the geometric algebra `g3`. Then `grad * F`, `grad | F`, and `grad ∧ F` are the gradient, divergence, and curl of F.

The directional derivative of F in the direction **a** is `(a | grad)*F`

Manifolds. This example creates the unit sphere sp2 in $\mathbb{R}^3$ as a submanifold of the geometric algebra g3 from Section 2.2:

```
sp2coords = (phi,theta) = symbols('phi theta', real=True)
sp2param = [sin(phi)*cos(theta), sin(phi)*sin(theta), cos(phi)]
```

Parameterize sp2 in terms of the x, y, z coordinates of g3

```
sp2 = g3.sm(sp2param, sp2coords)
(ephi, etheta) = sp2.mv()
```

Assign basis vector names for program.

Multivectors can be expressed in either the sp2 basis (ephi, etheta) for the tangent plane or the g3 basis (ex, ey, ez) for $\mathbb{R}^3$.

You can do petty much anything with the geometric algebra sp2 that you can with the geometric algebra g3. Examples:

```
f = sp2.mv('f','vector',f=True)
sp2grad = sp2.grad
```

Here is another way to create the unit sphere in $\mathbb{R}^3$, this time as a submanifold of the geometric algebra sp3 from Section 2.2:

```
sp2coords = (p,t) = symbols('p t', real=True)
```

p = phi, t = theta

```
sp2param = [1, p, t] # Parameterization of unit sphere
sp2 = sp3.sm(sp2param, sp2coords)
sp2grad = sp2.grad
```

4 Printing

$\mathcal{GA}$lgebra has two modes of output: to a console (terminal) or to a pdf with beautiful LaTeX typesetting.

Section 1 described the console ConEmu for Windows. For console output change ComEmu's default colors: Right click on its Title Bar/Settings/Colors. Then set Text: #0; Back: #7; Popup: #0; Back: #7; and ☑ Extend forground colors with background #13. If you find something that looks better to you, let me know.

Subscripts and superscripts. With the code

```
sp2coords = (phi,th) = symbols('phi theta', real=True)
```

the short "th" is used in the program, e.g., `print th`. However, and this is the point, the print statement sends "theta" to a console, and θ to a pdf.

Similarly, with the code

```
sp3 = Ga('e_r e_phi e_theta', ...  )
(er, ephi, eth) = sp3.mv()
```

the "eth" is used in the program, e.g., `print eth`. However, the print statement sends "e_theta" to a console and $\mathbf{e}_\theta$ to a pdf. **** submanifold bases?

The statement

```
print g2.mv('V', 'vector')
```

sends "V__x*e_x + V__y*e_y" to the console and $V^x\mathbf{e}_x + V^y\mathbf{e}_y$ to a pdf.

With console output, "\n " (note space) in a string in a print statement starts a new line.

Fmt. The command `Fmt(n)` specifies how multivectors are split over lines:

$n = 1$: The multivector is printed on one line. (The default.)

$n = 2$: Each grade of the multivector is printed on a separate line.

$n = 3$: Each component of the multivector is printed on a separate line

The $n = 2$ and $n = 3$ options are useful when a multivector will not fit on one line. The code `print A.Fmt(n)`, then A will print as specified. And `print A.Fmt(n,'B')` will print the string B = followed by A, as specified.

You can print a variable with one n and later with another.

Enhanced printing. Output is color coded for easier reading in console mode with enhanced printing. Issue these commands:

```
from printer import Eprint
Eprint() # right after the import statements
```

LaTeX output. In LaTeX mode, the pdf output file is automatically opened in a pdf reader when your program ends.[1] It is helpful, but not necessary, to know a bit of LaTeX for this. Of course you need a LaTeX system on your computer.[2] You need these statements to use LaTeX printing:

```
from printer import *
Format() # after imports
xpdf() # last statement of program.
```

In Linux, the output is opened in the standard *evince* pdf reader. In Windows, it is opened in the default pdf reader. (You must close the pdf reader, or at least the tab for your file, before rerunning your program. Otherwise your program will hang.) In OS X ????

Here is an example using LaTeX with $\mathcal{GA}$lgebra. When printing a string, an underscore "_" designates a subscript. A caret "^" (not a double underscore) designates a superscript. The statement

`print r'\alpha_1\bm{X}/\gamma_r^3'`

produces the output $\alpha_1\boldsymbol{X}/\gamma_r^3$. Note the r preceding the string. It prevents certain undesirable (from $\mathcal{GA}$lgebra's point of view) Python processing of strings with backslashes.

grad	∇
^	$\wedge$
$*$	
$\vert$	$\cdot$
$<$	$\rfloor$
$>$	$\lfloor$

If the string contains an "=", e.g., `r'XXX = YYY'`, then substitutions are made in `XXX` (only) according to the table. Thus

`r'grad A^B | * = grad A^B | *'`

prints as $\nabla A \wedge B\cdot = \text{grad} A^B|*$

A newline \n cannot appear in a string preceded by an r. Instead use `r'A' +'\n' + r'B'` to split `'AB'` into two lines. The +'s glue (concatenate) strings.

The parameters `Format(Fmode=True, Dmode=True)` give additional formatting options. Use them independently.

`Fmode=True`. Suppress function arguments: f, not the default $f(x,y)$.

`Dmode=True`. Use condensed partial derivative notation: $\partial_x f$, not the default $\frac{\partial f}{\partial x}$.

The file Symbols.pdf lists common LaTeX symbols.

[1]See GA.pdf for a fuller account.

[2]TeX Live is known to work, as is MiKTeX on Windows. I think that it is only necessary that the LaTeX system provide pdflatex. Please let me know if you find otherwise.

5 IPython Notebook

This section is not an IPython Notebook tutorial. (For one thing, I don't know enough to write one.) But it will help get you get started with the software. Please send me suggestions about ways to make it easier to use the notebook on Windows, Linux, and/or OS X.

Installation: `http://ipython.org/install.html`
Description and many links: `https://en.wikipedia.org/wiki/IPython`
Introduction: `http://ipython.org/notebook.html`
Quick tutorials:
  `http://ipython.org/ipython-doc/1/interactive/notebook.html`
  Another Click (The URL is too long.)
Documentation:
  `http://ipython.org/documentation.html`.
  `http://ipython.org/ipython-doc/stable/notebook/index.html`.
  Another Click.
Example presentations: `http://ipython.org/presentation.html`.

Execute. To execute a notebook cell, press shift+enter

Output. All output is typeset by LaTeX. Do not use `print`: Use `ex`, not `print ex`; use `A.Fmt()`, not `print A.Fmt()`. Do not use `Format()` or `xpdf()`.

Starting IPython Notebook. After installation, IPython.exe is in the Scripts folder of a Python installation. To start IPython in notebook mode, add notebook to its command line.

The notebook file extension is ipynb. On my Windows machine I created a bat file with a single line: "ipython notebook %1" (no quotes) and associated ipynb files to it.

I made several "get started" notebook files, one for each geometric algebra in Algebras.py (See Section 2.2). For example, g3.ipynb imports SymPy and $\mathcal{GA}$lgebra and sets up the geometric algebra g3. I can open the notebook and execute g3's cell, and be ready to make g3 calculations interactively. Better, make g3.ipynb read-only. Then when starting a new g3 notebook, copy g3.ipynb to MyNewNotebook.ipynb (not read-only) and use it for the notebook.

$\mathcal{GA}$lgebra's option to print to a pdf is not available in a notebook. However, it is possible to convert a Notebook to a Python program (and other formats): `http://ipython.org/ipython-doc/1/interactive/nbconvert.html`