Working code of RCC for E_6

Abhishek Parab

2/21/2018

```
%md
# Root Cone Conjecture for the root system E_6

**Background:**

My thesis is related to the [Arthur-Selberg Trace Formula](https://\
    en.wikipedia.org/wiki/Arthur%E2%80%93Selberg_trace_formula). I \
    have reduced the convergence of the geometric side of the twisted\
     trace formula to a geometric condition about root systems. If \
    the group $G$ is split, I prove this case by case, namely by \
    proving the Root Cone Conjecture holds when $G$ is a product of \
    simple split groups, and then proving for every split simple \
    group using the Cartan-Killing classification. For details, I \
    will refer to my thesis which is freely available here - [Arxiv\
    :1710.08885](https://arxiv.org/abs/1710.08885).

The Cartan-Killing classification says, the root system of a split \
    semisimple group is either one of the four infinite families, \
    $A_n, B_n, C_n, D_n$ or one of the exceptional groups $E_6, E_7, \
    E_8, F_4, G_2$. Among the exceptional groups however, only $E_6$ \
    has a nontrivial automorphism $\theta$, that of switching the \
    left and right sides. It's Weyl group has ~50,000 elements so the\
     code below performs a brute-force check for each element $w \in \
    W$ that a non-empty root cone exists. More precisely, the code \
    verifies that a point exists and by continuity of the equations, \
    we get a cone.

**Code below.**

The code written below in SageMath and Python and implements a brute\
     force algorithm to find a point (solution) to a set of \
    inequalities that characterizes the Root Cone Conjecture. It is \
    pretty self-explanatory.
```

1 Root Cone Conjecture for the root system E_6

Background:

My thesis is related to the Arthur-Selberg Trace Formula. I have reduced the convergence of the geometric side of the twisted trace formula to a geometric condition about root systems. If the group G is split, I prove this case by case, namely by proving the Root Cone Conjecture holds when G is a product of simple split groups, and then proving for every split simple group using the Cartan-Killing classification. For details, I will refer to my thesis which is freely available here - Arxiv:1710.08885.

The Cartan-Killing classification says, the root system of a split semisimple group is either one of the four infinite families, A_n, B_n, C_n, D_n or one of the exceptional groups E_6, E_7, E_8, F_4, G_2. Among the exceptional groups however, only E_6 has a nontrivial automorphism θ, that of switching the left and right sides. It's Weyl group has 50,000 elements so the code below performs a brute-force check for each element $w \in W$ that a non-empty root cone exists. More precisely, the code verifies that a point exists and by continuity of the equations, we get a cone.

Code below.

The code written below in SageMath and Python and implements a brute force algorithm to find a point (solution) to a set of inequalities that characterizes the Root Cone Conjecture. It is pretty self-explanatory.

```
# Proof of the root cone conjecture for the (unique) automorphism of\
    the Dynkin Diagram of E_6 using "Sage".

import time
start_time = time.time()
#R = RootSystem(['A', 2]);
R = RootSystem(['E', 6]);

# FiniteFamily was needed for the function theta(vector) earlier.
#from sage.sets.family import FiniteFamily

X = R.root_space()
alpha = X.basis()
alphacheck = X.coroot_space().basis()
varpi = X.fundamental_weights_from_simple_roots()
varpicheck = X.coroot_space().fundamental_weights_from_simple_roots\
    ()

#Function theta:
def theta(vector):
    sigma = PermutationGroupElement('(1,6)(3,5)(2)(4)')
    #sigma = PermutationGroupElement('(1,2)')
    for i in range(1, len(varpi)+1):
        if vector == varpi[i]:
            break
    return varpi[sigma(i)]

# Function delta:
def delta(w):
```

```
    list = []
    for item in varpi:
        if (w.action(item)).to_ambient() != item.to_ambient():
            list.append(item)
    return list
#for w in W:
#    print "For w = \n", w, "the set Delta(w) is ", delta(w)

# Function returning the vector varpi_i - w theta varpi_i in the \
   ambient space:
def rhs(w, vector):
    return (vector - w.action(theta(vector))).to_ambient()

# Function that returns True if < lambda, varpi - w theta varpi> is \
   positive
def isPositive(Lambda, w):
    for vector in delta(w):
        if (Lambda.to_ambient()).dot_product(rhs(w, vector)) <= 0:
            return False
    return True

# Function returning True if a vector Lambda is found for the \
   element w
def isSuccess(w):
    for x_1 in range(1,3):
        for x_2 in range(1,3):
            for x_3 in range(1,3):
                for x_4 in range(1,3):
                    for x_5 in range(1,3):
                        for x_6 in range(1,3):
                            Lambda = x_1 * varpicheck[1] + x_2 * \
   varpicheck[2] + x_3 * varpicheck[3] + x_4 * varpicheck[4] + x_5 *\
    varpicheck[5] + x_6 * varpicheck[6]
                            if isPositive(Lambda, w):
                                return (Lambda, True)
    Lambda = 0 * varpicheck[1]
    return (Lambda, False)

#Note to self: I can access the roots, coroots, weights and \\
   ncoweights by alpha, alphacheck, varpi, varpicheck.

W = X.weyl_group()
w = W.an_element()
# Testing:
# print "An element of the Weyl group is\n", w
# print "It's action on alpha_1 and alpha_2 is respectively,", w.\
   action(alpha[1]), "and ", w.action(alpha[2])
```

```
# print "It's action on varpi_1 and varpi_2 is respectively,", w.\
   action(varpi[1]), "and ", w.action(varpi[2])
# print "It doesn't act on the coroots and coweights."
#w.action(alpha[1]) == -1*varpi[1] + 2 * varpi[2]     # works fine!
count = 0
for w in W:
    if isSuccess(w)[1] == True:
        count = count + 1
    else:
        print "Root Cone Conjecture fails for w = \n", w
print "Number of successful elements = ", count, "(Order of W =", W.\
   cardinality(), ")"
print "\n(Time to run code: %s seconds)" % (time.time() - start_time\
   )
```

(Time to run code: 0.0925168991089 seconds)