README.md

Caspar

A Python implementation of the SU(n) factorization scheme of [citation forthcoming].

Dependencies

Caspar is written in Python 3. You will also need the numpy package.

Installation

In the main directory of the folder, type `python setup.py install`.

Usage (basic)

There are two important files: `factorization_script.py`, and `user_matrix.py`. Enter the SU(n) matrix you wish to factorize in the variable `SUn_mat` in `user_matrix.py`. Then, factorize it by running

```
python factorization_script.py
```

The output of this script will be a series of lines in the following format, for example:

```
4,5    [-2.8209, 2.5309, 2.3985]
3,4    [-1.7534, 1.4869, -1.753]
...
```

This is a sequence of SU(2) transformations. The first two integers indicate the modes on which the transformation acts. The set of three floats are the parameters of the transformation (see parametrization below). The original matrix `SUn_mat` is obtained by embedding each SU(2) transformation into the indicated modes of an SU(n) transformation, and multiplying them together from top to bottom of the list (with each transformation added to the product on the right, e.g. $U = U_{45}\, U_{34}...$).

Important note: at the current time, Caspar works well for up to about n = 11. After this point, it begins to experience numerical issues due to very small numbers.

Usage (detailed)

An arbitrary element of SU(n) can be fully expressed using at most n^2 - 1 parameters. We put forth a factorization scheme that decomposes elements of SU(n) as a sequence of SU(2) transformations. SU(2) transformations require in general 3 parameters, [a, b, g], written in matrix form as [[$e^{i(a+g)/2}$ cos(b/2), -$e^{i(a-g)/2}$ sin(b/2)], [$e^{-i(a-g)/2}$ sin(b/2), $e^{i(a+g)/2}$ cos(b/2)]].

There are two main functions: `sun_factorization` and `sun_reconstruction`, each contained in the appropriately named files.

The function `sun_factorization` takes an SU(n) matrix (as a numpy array) and decomposes it into a sequence of n(n-1)/2 such SU(2) transformations. The full set of n^2 - 1 parameters is returned as a list of tuples of the form ("i,i+1", [a_k, b_k, g_k]) where i and i+1 indicate the modes on which the transformation acts (our factorization uses transformations only on adjacent modes).

The following code snippet can be used to factorize the SU(3) matrix below.

```
import numpy as np
from caspar import sun_factorization

n = 3

SUn_mat = np.array([[0., 0., 1.],
                    [np.exp(2 * 1j * np.pi/ 3), 0., 0.],
                    [0., np.exp(-2 * 1j * np.pi / 3), 0.]])

# Perform the decomposition
parameters = sun_factorization(SUn_mat)

# The output produced is
#
# Factorization parameters:
#   2,3    [2.0943951023931953, 0.0, 2.0943951023931953]
#   1,2    [0.0, 3.1415926535897931, 0.0]
#   2,3    [0.0, 3.1415926535897931, 0.0]
```

It is also possible to reconstruct an SU(n) transformation based on a list of parameters for SU(2) transformations given in the form ("i,i+1", [a_k, b_k, g_k]). The matrix is computed by multiplication on the right. At the moment only adjacent mode transformations are supported.

```
from caspar import sun_reconstruction

parameters = [("1,2", [1.23, 2.34, 0.999]),
              ("4,5", [0.3228328, 0.23324, -0.2228])]

new_SUn_mat = sun_reconstruction(6, parameters)
```