REPO LINK: https://github.com/neal-p/CHEM279

HW2

Compilation Instructions

1. Go to the top level repo directory `CHEM279/`
2. `mkdir -p build`
3. `cd build`
4. `cmake ..`
5. `make hw2_1 hw2_2`

Run Instructions

Once you have compiled, there will be two executables: - `CHEM279/HW2/hw2_1` - `CHEM279/HW2/hw2_2`

These correspond to each of the two problems.

Problem 1

Here, the `hw2_1` executable will read a provided 1D gaussian functions from the input file storing them as a `vector` of `Gauss` objects. Then, it will numerically integrate the product to yield the overlap.

I implemented `qsimp` from the Numerical Recipies book (page 165) which is an improvement on the basic trapezoid rule, which is a version of Simpson's rule that is 'more efficient than [trapezoid rule]' and is 'a good one for light-duty work'. Which I felt very much fits the desire for efficiency, accuracy, but not overly complex.

The `Gauss` class, and another class `GaussProd` (which is the product of two `Gauss` instances used for the integration), extend an abstract base class `Integrand`, which is what the `QSIMP` integration method takes as an argument. This makes the integration method able to accept any other instances of `Integrands` for future re-use.

The `sample_input/` folder contains the provided sample input. You can run these tests by executing `run_samples_problem1.sh`, which will run `./hw2_1 <file> > my_output/numerical/<file>` for each sample and pipe the output to a file in `my_output`.

An example output is shown below:

```
./hw2_1 sample_input/numerical/1.txt

read gaussian centered at 0 with alpha=1 and l=0
read gaussian centered at 0 with alpha=1 and l=0
1d numerical overlap integral between Gaussian functions is 1.25331
```

Problem 2

Here, the `hw2_2` executable will read the provided 3D orbitals defined by the input file storing them as a `vector` of `Shell` objects. This shell object will enumerate the required angular momentum terms for a set of primative gaussians to describe the shell.

Here, I've been less modular in the code than usual. While I created an abstract base class and made my numerical integration modular, here I've chosen to write a specific function `ShellOverlap` that takes two `Shell`s and analytically integrates them. What I loose in modularity, I gain in some efficiency in evaluating the integral. I was able to pull apart the angular-momentum and non-angular-momentum terms to prevent re-calculation. If instead I wrote a general 1D overlap function and called it many times within `ShellOverlap` then I would end up re-computing all the non-angular momentum terms each iteration of the loop. My solution also heavily leverages vectorized calculations, using arary broadcasting on `Eigen::Array3d` arrays as much as possible. This is especially useful here being able to work on the xyz dimensions all at once in the array format.

There certainly is a middle ground between the specific analytical function I wrote and a more modular design. I will re-think if there is a clever way I can write the interface to get the best of both worlds, but for now this is my solution.

The `sample_input/` folder contains the provided sample input.

You can run these tests by executing `run_samples_problem2.sh`, which will run `./hw2_2 <file> my_output/analytical/<file.csv> > my_output/analytical/<file>` for each sample and pipe the output to a file in `my_output`.

An example output is shown below:

```
./hw2_2 sample_input/analytical/3.txt
```

```
Shell:
  alpha=1
  xyz=0 0 0
  L=2
  Composed of:
        lmn: 2 0 0
        lmn: 1 1 0
        lmn: 1 0 1
        lmn: 0 2 0
        lmn: 0 1 1
        lmn: 0 0 2


Shell:
  alpha=0.5
```

```
  xyz=1 1 0
  L=1
  Composed of:
        lmn: 1 0 0
        lmn: 0 1 0
        lmn: 0 0 1


Overlap matrix:
-0.115272 -0.461089         0
0.0576361 0.0576361         0
        0        -0  0.172908
-0.461089 -0.115272         0
       -0         0  0.172908
-0.345817 -0.345817         0
```