1. Dual numbers and partial derivatives

Extend the `dual.py` code we wrote in class to use dual numbers to automatically evaluate both partial derivatives of the following function $f(x, y)$:

```python
def f(x, y):
    return -2.0 * x * exp(-(x**2 + y**2)/(4.0))
```

You should evaluate the function and its derivatives at $(x, y) = (0.5, 1.5)$.

Here are two possible strategies:

- **Easier**: Call the function $f$ twice, each time with one `Dual` object as an argument, and one normal floating point number.

- **Harder**: Extend the `Dual` class so that `self.eps` is a flat `numpy` array of length two where the infinitesimal parts correspond to the partials in $x$ and $y$.

In either case, you will need to provide a definition of the `exp` function which handles dual numbers correctly.

Write a test function to show that your program computes the correct derivatives by comparing to the numerical central difference approximations for the first derivative. Submit a printout of your program and its output.
2. Solving the steady-state heat equation

The `steadystate_heat.py` code on the course website solves the steady-state heat equation with given boundary conditions that we defined in class.

a. Comment the code extensively so that it is clear to the reader what each part of the program is doing. Your comments should also address these questions:

- In particular, what are the `BOUNDARY.*` variables doing?
- How are they related to the `boundaries` array?
- What does `scipy.sparse.dok_matrix` do?
- What are the `neighbor_deltas` for?
- How the heck does the vectorized building code work?
- What does the line `A = A.tocsr()` do?

b. For what values of `n` is it advantageous to compute the ’dense’ solution as opposed to the ’sparse’ one? Is there any problem size for which the ’loop’ building method outperforms the ’vectorized’ one? Why is the sparse solver so much faster for large `n`?

Please submit a printout of your commented code, along with a brief writeup for part b above.