Math 260, Fall 2020
*Last updated: 10/18/20*

# HOMEWORK 8

DUE FRI. OCT. 23

**Note:** For details on sparse matrices that may be useful, see the documentation for `sparse.csc_matrix`, `sparrse.csr_matrix` and `sparse.coo_matrix` on scipy (you can google the name plus "scipy"). Midway through the page is a "Notes" section with advantages and disadvantages.

---

**Q1 (PageRank, for real this time).**
a) The power method needs the 'PageRank' matrix $M$, which we do not want to construct. Modify the stationary distribution code (the power method) so that it instead takes the transpose of the transition matrix ($P^T$) and the teleport parameter $\alpha$.

The matrix $P^T$ must be a sparse matrix; make sure you do not construct a full matrix!

b) Write a function that takes the data from your 'read graph from file' code and returns the sparse matrix $P^T$ (you'll also need the names for display later).

c) Finally, write a test that calculates the PageRank of the supplied graph, which is a list of URLs connected to a certain university website.[1]

Have your code output the top ten websites (in order) and their PageRank score.

**Remark:** It may be necessary to normalize the vector at each step (this is labeled as 'optional' in some example code, but needed here if there are dead end links).

---

**Q2.** Implement either the RK4 method or the explicit trapezoidal rule (your choice) for scalar ODEs $y' = f(t, y)$. Build a test case (e.g. use one of the existing ones from lecture) and verify that it has the correct order.
Your code should generate a clear convergence plot.

---

**Q3 (circles, again).** We saw outward spirals when Euler's method was used to solve

$$x' = y, \quad y' = -x$$

a) Derive an iteration for an approximation $(\tilde{x}_n, \tilde{y}_n)$ that uses the trapezoidal rule for each equation, e.g. for the $x$-component,

$$x_{n+1} = x_n + \int_{t_n}^{t_{n+1}} y(s)\, ds \implies \tilde{x}_{n+1} = \tilde{x}_n + \cdots .$$

b) Rearrange so you can compute $\tilde{x}_{n+1}$ and $\tilde{y}_{n+1}$ directly from the values at $t_n$ (this is an example where you can solve for the next step in an implicit method).

c) Implement this and use it to solve the ODE (note: you'll need to write code for this specific problem, rather than use code for a general ODE function $F(\mathbf{x})$). Compare to Euler's method - how is this approximation (qualitatively) different?

[1] Source: http://www.cs.cornell.edu/courses/cs685/2002fa/