Problem set 7

[Revised Feb 03, 9:48 PM]

Linear algebra

1. Write a Python funtion called `mat_vec` which computes the matrix vector product

   $$\mathbf{y} = \mathbf{A}\mathbf{x},$$

   where $\mathbf{A} \in \mathbb{R}^{M \times N}$, $\mathbf{y} \in \mathbb{R}^{M}$ and $\mathbf{x} \in \mathbb{R}^{N}$.

   Note that the i^{th} element of $\mathbf{y}$ is given by

   $$y_i = \sum_{j=1}^{N} a_{ij} x_j.$$

   - You should complete the function provided in `mat_vec.py`. To test your code, execute `python3 mat_vec.py`
   - The code provided uses the following test data

   ```
   x = np.array( [1.1, 2.2, 3.3] )
   ```

   ```
   A = np.array( [[1.0, 2.0, 3.0], [4.0, 5.0, 6.0]] )
   ```

2. Write a Python funtion called `mat_mult` which computes the matrix-matrix product

   $$\mathbf{A} = \mathbf{B}\mathbf{C},$$

   where $\mathbf{B} \in \mathbb{R}^{M \times N}$, $\mathbf{C} \in \mathbb{R}^{N \times P}$ and $\mathbf{A} \in \mathbb{R}^{M \times P}$.

   Note that the ij^{th} element of $\mathbf{A}$ is given by

   $$a_{ij} = \sum_{k=1}^{N} b_{ik} c_{kj}$$

- You should complete the function provided in `mat_mult.py`. To test your code, execute `python3 mat_mult.py`
- The code provided uses the following test data

```
B = np.array( [[1.0, 2.0, 3.0],  \
               [4.0, 5.0, 6.0]] )
```

```
C = np.array( [[1.0, 2.0, 3.0],  \
               [4.0, 5.0, 6.0],  \
               [7.0, 8.0, 9.0]] )
```

3. In this question we use the following matrix

```
A = numpy.array([[1.0, -2.0, 0.0],  \
                 [-2.0, 1.0, -2.0], \
                 [0.0, -2.0, 1.0]])
```

and vector

```
b = numpy.array([1.1, 2.2, 3.3])
```

- Compute the LU factorization of `A` using `scipy.linalg.lu_factor()`.
- Use `scipy.linalg.lu_solve()` to solve $\mathbf{Ax} = \mathbf{b}$ where `b` is given above. Store the solution for $\mathbf{x}$ in the variable `x0`.
- Use `numpy.linalg.solve()` with the `A` matrix and `b` vector (given above). Store the solution in the variable `x1`.
- Verify by computing the maximum difference between `x0` and `x1` that the two solutions are identical.

4. The file `create_tridiag.py` contains functions to create tri-diagonal matrices assembled as dense matrices (`create_mat()`) and sparse matrices (`create_sparse_mat()`). We wish to compare the solve time associated with using LU with a dense matrix format and a sparse matrix format

We wish to assemble the following triadiagonal matrix

$$
\mathbf{A} = \begin{bmatrix} 1 & -2 & 0 & \dots & \dots & 0 \\ -2 & 1 & -2 & 0 & 0 & \dots \\ 0 & -2 & 1 & -2 & 0 & \dots \\ \vdots & & \ddots & \ddots & \ddots & 0 \\ \vdots & \dots & 0 & -2 & 1 & -2 \\ 0 & \dots & \dots & 0 & -2 & 1 \end{bmatrix}
$$

for different size matrices $\mathbf{A} \in \mathbb{R}n \times n$.

For the dense solver use `xd = numpy.linalg.solve(Ad, b)`. For the sparse solver use `xs = scipy.sparse.linalg.spsolve(As, b)`, where `Ad` is the dense matrix and `As` is the sparse tri-diagonal matrix.

- Use `create_mat()` to assemble a sparse 3×3 tri-diagonal matrix with 1 on the diagonal and -2 on the off-disagonal. Print the solution and verify it is correct.
- Use `create_sparse_mat()` to assemble a 3×3 tri-diagonal matrix with 1 on the diagonal and -2 on the off-disagonal. Store the result in `As`. Print the solution and verify it is correct using `print(As.toarray())`.
- Time how long it takes to solve each $\mathbf{Ax} = \mathbf{b}$ with dense and sparse matrices when $\mathbf{A}$ is given by an $n \times n$ matrix and `n` is given by `n = [6000, 7000, 8000, 9000, 10000]`. Use `b = numpy.ones(n[i])`.
- Plot the solve time (y-axis) requried to solve $\mathbf{Ax} = \mathbf{b}$ using sparse and dense matrices as a function of n (x-axis)

You can time Python code using the following

```
import time

t_start = time.perf_counter()

# SOLVE Ax = b HERE

t_end = time.perf_counter()
time_solve = t_end - t_start
```