Intensive Computation

1st april 2014

Exercise 1
Write a script that:

- takes in input **n**, number of rows and columns, and **s,** sparsity
- creates the sparse matrices A and B, nxn, with n>10, with sparsity s, consisiting of random values in the interval [1,10]
- calls functions **toCSR** and **toCSC** that produce the CSR and CSC compact representation
- calls function **extractCol** that extracts a column from the CSR representation of B
- computes the product **C-CSR**= A-CSR*B-CSR
- calls function **extractRow** that extracts a row from the CSC representation of A
- computes the product **C-CSC**= A-CSC*B-CSC
- computes the product **C-RC**= A-CSR*B-CSC
- compares the execution times to obtain **C-CSR, C-CSC, C-CR,** using `cputime, etime, tic,…, toc`.

Exercise 2
Write a script that:

- takes in input: **n**, number of rows and columns, **k**, the parameter for the **bandwidth b**, with b=2k+1 (in other words, k is the number of diagonals under, or over, the main diagonal), and **s,** sparsity inside the band
- creates the band sparse matrices A and B, nxn, with n>10, k>n/3, and sparsity s, consisiting of random values in the interval [1,10]
- calls function **toSkyline** that produces the Skyline compact representation
- computes the product **C-Sky**= A-Sky*B-Sky

Sparse Matrices in Matlab

S=sparse(A) converts a full matrix to sparse form by squeezing out any zero elements. If `S` is already sparse, `sparse(S)` returns `S`.

S = sparse(i,j,s,m,n,nzmax) uses vectors `i`, `j`, and `s` to generate an `m`-by-`n` sparse matrix with elements vector `s` with indices in vectors `i` and `j`, such that `S(i(k),j(k)) = (k)`, with space allocated for `nzmax` nonzeros. Vectors `i`, `j`, and `s` are all the same length.

A=full(S) converts a sparse matrix `S` to full storage organization. If `S` is a full matrix, then `A` is identical to `S`.

Example:

```
>> x =[5 9 1 7 3]
>> S=sparse ([2 4 1 3 6] ,[1 1 3 3 7],x)
S=
(2,1) 5
(4,1) 9
(1,3) 1
(3,3) 7
(6,7) 3

>> full(S)
ans =
0 0 1 0 0 0 0
5 0 0 0 0 0 0
0 0 7 0 0 0 0
9 0 0 0 0 0 0
0 0 0 0 0 0 0
0 0 0 0 0 0 3
```

Matlab includes many commands for dealing with a sparse matrix:

nnz(A) returns the number of nonzero matrix elements
nzmax(A) returns the maximum number of nonzero matrix elements allocated
find(A) returns all (i,j) indices of nonzero elements
nonzeros(A) returns all the nonzero elements

Spy(S) plots the sparsity pattern of any matrix `S`

R = sprand(m,n,density) is a random, `m`-by-`n`, sparse matrix with approximately `density*m*n` uniformly distributed nonzero entries (`0 <= density <= 1`)

A=spdiags(b,d,m,n) creates an `m`-by-`n` sparse matrix by taking the columns of `B` and placing them along the diagonals specified by `d`.

Example:

```
>> n=10;
>> e=ones(n,1);
>> b=[e,-e,3*e,-e,2*e];
>> d=[-n/2 -1 0 1 n/2];
>> a=spdiags(b,d,n,n)
a =
(1,1) 3
(2,1) -1
(6,1) 1
(1,2) -1
(2,2) 3
(3,2) -1
……
```

```
>> aa=full(a)
aa =
 3 -1  0  0  0  2  0  0  0  0
-1  3 -1  0  0  0  2  0  0  0
 0 -1  3 -1  0  0  0  2  0  0
 0  0 -1  3 -1  0  0  0  2  0
 0  0  0 -1  3 -1  0  0  0  2
 1  0  0  0 -1  3 -1  0  0  0
 0  1  0  0  0 -1  3 -1  0  0
 0  0  1  0  0  0 -1  3 -1  0
 0  0  0  1  0  0  0 -1  3 -1
 0  0  0  0  1  0  0  0 -1  3
```

Example of tridiagonal matrix:

```
>> b=ones(4,1);
>> A=spdiags([b 3*b b],-1:1,4,4)
A =
(1,1) 3
(2,1) 1
(1,2) 1
(2,2) 3
(3,2) 1
(2,3) 1
(3,3) 3
(4,3) 1
(3,4) 1
(4,4) 3
```

```
>> d=full(A)
d =
3 1 0 0
1 3 1 0
0 1 3 1
0 0 1 3
```

Example: comparison of memory occupation

```
>> b=ones(100,1);
>> A=spdiags([b 3*b b],-1:1,100,100)
>> d=full(A);

>> whos
Name      Size            Bytes     Class
A         100x100         3980      double array (sparse)
b         100x1           800       double array
d         100x100         80000     double array
```

Example: comparison of execution time needed to compute the square of a matrix in the full and in the sparse representation

```
>> a=eye(1000);
>> t=cputime;
>> b=a^2;
>> temp=cputime-t
temp =
3.7454

>> a=sparse(1:1000,1:1000,1,1000,1000);
>> t=cputime;
>> c=a^2;
>> temp=cputime-t
temp =
0.4406
```

`gplot(A,Coordinates)` plots a graph of the nodes defined in `Coordinates` according to the *n*-by-*n* adjacency matrix `A`, where *n* is the number of nodes. `Coordinates` is an *n*-by-2 matrix, where *n* is the number of nodes and each coordinate pair represents one node.

Example
One interesting construction for graph analysis is the *Bucky ball*. This is composed of 60 points distributed on the surface of a sphere in such a way that the distance from any point to its nearest neighbors is the same for all the points. Each point has exactly three neighbors. The Bucky ball models different physical objects, such as the C_{60} molecule, a form of pure carbon with 60 atoms in a nearly spherical configuration and the seams in a soccer ball

```
      [B,v]=bucky; % B= adjacency matrix, v= coordinate matrix
      gplot(B,v)
      axis square
------
      [B,v]=bucky;
      axis('square');hold on
      gplot(B(1:30,1:30),v)
      for k=1:30
      text(v(k,1),v(k,2),num2str(k))
      end
```