

CS 550: Iterative and Multigrid Methods
Spring 2007

Homework, Set 1

Due Tuesday February 6, 2007

Write descriptive solutions. Comment your code!

The goal of this homework set is to build a basic sparse matrix package. You have the option to complete this task in C, C++, or a hybrid. The following write-up assumes a C approach. C++ comments will be noted at the end.

To initialize the sparse package you will need the following principle sections:

- `vector.h`: contains the vector struct and vector "member" functions
- `matrix.h`: contains the sparse matrix structs, conversions, and matrix "member" functions
- `splat.h`: sparse linear algebra techniques (I,II,II)
- `util.h`: utilities to generate a laplacian, matrix-market interface (mmio)
- (later) `basicsolve.h`: jacobi, gauss-seidel, sor, MR, SD, etc
- (later) `krylovsolve.h`: CG, GMRES, variants, etc.
- (later) `tkrylovsolve.h`: BCG, QMR, etc
- (later) `multigrid.h`: GMG, AMG

We will focus on starting the first four.

`vector.h` Build a double vector struct (or class) that contains the length (n) and value array *val*. For example:

```
typedef struct {
    int n;
    double *val;
} DVec;
```

`vector.h` Create a constructor that accepts n and initializes to 0, a destructor that destroys the vector, a print routine that prints the vector to an ASCII file, and several copy constructors. For example:

```
DVec *DVec_new0(int n);
void DVec_delete(DVec *v);
void DVec_print(DVec *v);

void Dvec_copyvec(DVec *v, DVec *u); /* v = u */
void Dvec_copydouble(DVec *v, double *u); /* v = u */
void Dvec_copyconst(DVec *v, const double u); /* v[i] = u */
```

`matrix.h` Use a struct to define CSR, COO, and LL type matrices. You may wish to include a `matrixtype` to distinguish between symmetric and unsymmetric versions for future use. For example:

```
typedef enum { SYMMETRIC, UNSYMMETRIC } matrixtype;

typedef struct {
    matrixtype type;
    int m, n, nnz;
    double *val;
```

```

    int *row, *col;
} CSRMat, CSCMat, COOMat;

typedef struct {
    matrixtype type;
    int m, n, nnz;
    double *val;
    int *col, *next, *root;
} LLMat;

```

matrix.h Similar to vector, create a new (which simply allocates memory and does not assign a structure, delete, and print routine for these matrices. For example:

```

CSRMat *CSRMat_new    (int m, int n, int nnz);
void    CSRMat_delete(CSRMat *A);
void    CSRMat_print  (CSRMat *A);

```

The print routines can call the mmio package.

matrix.h For now, we can rely on limited access. Build a get method for CSR and get/set methods for COO and LL formats. For example:

```

double CSRMat_get(CSRMat *A, int i, int j);
double COOMat_get(COOMat *A, int i, int j);
double LLMat_get(LLMat *A, int i, int j);
void COOMat_set(COOMat *A, int i, int j, double a);
void LLMat_set(LLMat *A, int i, int j, double a);

```

matrix.h For starters, create a conversion routine from LL to CSR and COO to CSR. For example:

```

CSRMat *COO2CSR (COOMat *A);
CSRMat *LL2CSR (COOMat *A);

```

splat.h We'll build only the basic linear algebra subroutines that are needed for our iterative methods. For example:

```

/* level 1 */
double dot(Vec *u, Vec *v);           /* a = dot(u,v) */
double norm(Vec *u);                  /* a = ||u|| */
void scalevec(Vec *u, Vec *v, const double *a); /* v = a*u */
void add(const double a, Vec *u,
         const double b, Vec *v);     /* v = a*u +b*v */

/* level 2 */
void scalemat(CSRMat *A, const double *a); /* A = aA */
void matvec(CSRMat *A, Vec *x, double *a, Vec *y); /* y = Ax + by */
void res(CSRMat *A, Vec *x, Vec *b, Vec *r); /* r= b-Ax calls
    matvec */
void normres(CSRMat *A, Vec *x,
            Vec * b, double *b); /* a= ||b-Ax|| calls matvec + norm */

/* level 3 */
void matmat(CSRMat *A, CSRMat *B, CSRMat *C); /* C=A*B */

```

Later, we'll need a galerkin product, but this should be sufficient for now.

`util.h` Finally, create an interface, `genA`, to call the FDIF and FEM utilities in Saad's SPARSEKIT2. The return matrix A should be in CSR format. Be careful with the 1-based indexing of FORTRAN. You may want to put an interface to `mmio.c` and `mmio.h` files from the Matrix Market also in this location.

Comments: While this HW set contains many prototypes, the actual code is quite short. The lengthier routines will be the conversion, set, and mat-mat functions. You can easily find all of these routines in many other packages (bebop, sparsekit2, blitz, CSparse, gsl, ITSOL, NIST spblas, sl, spai, sparselib, splib, and many many more). I expect you to reference, rip, copy, follow these codes where applicable (and legal). In the end you should have your own implementation, but please do not hesitate to get direction from these other codes.

Also, the code outlined above is merely a suggestive naming convention and style. Rename and modify this template as you see fit. The C++ implementation of classes and member functions should be clear. One piece of advice in either C or C++, but particularly in C++, is to be careful of the temporary storage. The copy constructor in C++ is often a problem particularly when overloading arithmetic operators (see Stroustrup p. 675). Another common hang-up is the difference between a *shallow copy* and a *deep copy*; it may be helpful to review this terminology.

Lastly, grading for the HW will look at functionality. I do not expect error handling, templating (in C++), abstractions, or fancy techniques. The goal is to build a functional sparse matrix package. As I am not providing a specific API, I do expect a `test.c` (and accompanying `Makefile`) to verify your assortment of routines. It need not be an exhaustive test suite, but should validate your routines on some level.

Please hand in a tarred-gzipped file with the following name:

`CS5500_hw1_FirstInitialLastName.tgz`

For example:

`CS5500_hw1_L01son.tgz`

This should unpack in the directory

`CS5500_hw1_L01son`

with the command

`tar xvfz CS5500_hw1_L01son.tgz`

Comments or homework write-up should be included in this directory under `hw1.pdf`.