Chapter 1

A2: Real or complex 2-D array

The A2 object is one way to store and operate on and with a dense matrix. The matrix can contain either double precision real or complex entries. It is used as a lower level object for the DenseMtx object, and during the QR factorization to hold a staircase matrix.

1.1 Data Structure

The A2 structure has six fields.

- `int type` : type of entries, SPOOLES_REAL or SPOOLES_COMPLEX
- `int n1` : size in first dimension, number of rows
- `int n2` : size in second dimension, number of columns
- `int inc1` : increment or stride in first dimension
- `int inc2` : increment or stride in second dimension
- `int nowned` : the number of entries that are "owned" by this object. When `nowned > 0`, `entries` points to storage for `nowned` entries, (`nowned double`'s for the real case, `2*nowned double`'s for the complex case), that have been allocated by this object and can be free'd by the object. When `nowned == 0` but `n1 > 0` and `n2 > 0`, this object points to entries that have been allocated elsewhere, and these entries will not be free'd by this object.
- `double *entries` : pointer to the base address of the *double* vector

One can query the properties of the front matrix object using these simple macros.

- `A2_IS_REAL(mtx)` is `1` if `mtx` has real entries and `0` otherwise.
- `A2_IS_COMPLEX(mtx)` is `1` if `mtx` has complex entries and `0` otherwise.

The `A2_copyEntriesToVector()` method uses the following constants: `A2_STRICT_LOWER`, `A2_LOWER`, `A2_DIAGONAL`, `A2_UPPER`, `A2_STRICT_UPPER`, `A2_ALL_ENTRIES`, `A2_BY_ROWS` and `A2_BY_COLUMNS`.

1.2 Prototypes and descriptions of A2 methods

This section contains brief descriptions including prototypes of all methods that belong to the A2 object.

1.2.1 Basic methods

As usual, there are four basic methods to support object creation, setting default fields, clearing any allocated data, and free'ing the object.

1. `A2 * A2_new ( void ) ;`

   This method simply allocates storage for the `A2` structure and then sets the default fields by a call to `A2_setDefaultFields()`.

2. `void A2_setDefaultFields ( A2 *mtx ) ;`

   The structure's fields are set to default values: `type` = `SPOOLES_REAL`, `n1` = `inc1` = `n2` = `inc2` = `nowned` = 0 and `entries` = `NULL` .

   Error checking: If `mtx` is `NULL`, an error message is printed and the program exits.

3. `void A2_clearData ( A2 *mtx ) ;`

   This method clears the object and free's any owned data. If `nowned > 0` and `entries` is not `NULL`, then `DVfree(entries)` is called to free the storage. It calls `A2_setDefaultFields()`.

   Error checking: If `mtx` is `NULL`, an error message is printed and the program exits.

4. `void A2_free ( A2 *mtx ) ;`

   This method releases any storage by a call to `A2_clearData()` and then free the space for `mtx`.

   Error checking: If `mtx` is `NULL`, an error message is printed and the program exits.

1.2.2 Instance methods

1. `int A2_nrow ( A2 *mtx ) ;`

   This method returns the number of rows in the matrix.

   Error checking: If `mtx` is `NULL`, an error message is printed and the program exits.

2. `int A2_ncol ( A2 *mtx ) ;`

   This method returns the number of columns in the matrix.

   Error checking: If `mtx` is `NULL`, an error message is printed and the program exits.

3. `int A2_inc1 ( A2 *mtx ) ;`

   This method returns the primary increment, the stride in memory (with respect to real or complex entries) between adjacent entries in the same column.

   Error checking: If `mtx` is `NULL`, an error message is printed and the program exits.

4. `int A2_inc2 ( A2 *mtx ) ;`

   This method returns the secondary increment, the stride in memory (with respect to real or complex entries) between adjacent entries in the same row.

   Error checking: If `mtx` is `NULL`, an error message is printed and the program exits.

5. `double * A2_entries ( A2 *mtx ) ;`

   This method returns a pointer to the base address of the entries.

   Error checking: If `mtx` is `NULL`, an error message is printed and the program exits.

6. `double * A2_row ( A2 *mtx, int irow ) ;`

   This method returns a pointer to the leading element of row `irow`.

   Error checking: If `mtx` or `entries` is `NULL`, or if `irow` is not in `[0,n1-1]`, an error message is printed and the program exits.

7. `double * A2_column ( A2 *mtx, int jcol ) ;`

   This method returns a pointer to the leading element of column `jcol`.

   Error checking: If `mtx` or `entries` is `NULL`, or if `jcol` is not in `[0,n2-1]`, an error message is printed and the program exits.

8. `void A2_realEntry ( A2 *mtx, int irow, int jcol, double *pValue ) ;`

   This method fills `*pValue` with the entry in location `(irow, jcol)`.

   Error checking: If `mtx` or `pValue` is `NULL`, or if the matrix is not real, or `irow` is not in `[0,n1-1]`, or if `jcol` is not in `[0,n2-1]`, an error message is printed and the program exits.

9. ```
   void A2_complexEntry ( A2 *mtx, int irow, int jcol,
                          double *pReal, double *pImag ) ;
   ```

   This method fills `(*pReal,*pImag)` with the entry in location `(irow, jcol)`.

   Error checking: If `mtx`, `pReal` or `pImag` is `NULL`, or if the matrix is not complex, or `irow` is not in `[0,n1-1]`, or if `jcol` is not in `[0,n2-1]`, an error message is printed and the program exits.

10. `void A2_setRealEntry ( A2 *mtx, int irow, int jcol, double value ) ;`

    This method sets entry `(irow,jcol)` to `value`.

    Error checking: If `mtx` is `NULL`, or if the matrix is not real, or `irow` is not in `[0,n1-1]` or if `jcol` is not in `[0,n2-1]`, an error message is printed and the program exits.

11. ```
    void A2_setComplexEntry ( A2 *mtx, int irow, int jcol,
                              double real, double imag ) ;
    ```

    This method sets entry `(irow,jcol)` to `(real,imag)`.

    Error checking: If `mtx` is `NULL`, or if the matrix is not complex, or `irow` is not in `[0,n1-1]` or if `jcol` is not in `[0,n2-1]`, an error message is printed and the program exits.

12. `void A2_pointerToRealEntry ( A2 *mtx, int irow, int jcol, double **ppValue ) ;`

    This method sets `*ppValue` to the pointer of the `(irow,jcol)` entry.

    Error checking: If `mtx` or `ppValue` is `NULL`, or if the matrix is not real, or if `irow` is not in `[0,n1-1]`, or if `jcol` is not in `[0,n2-1]`, an error message is printed and the program exits.

13. ```
    void A2_pointerToComplexEntry ( A2 *mtx, int irow, int jcol,
                                    double **ppReal, double **ppImag ) ;
    ```

    This method sets `*ppReal` to the pointer to the real part of the `(irow,jcol)` entry, and sets `*ppImag` to the pointer to the imaginary part of the `(irow,jcol)` entry.

    Error checking: If `mtx`, `ppReal` or `ppImag` is `NULL`, or if the matrix is not complex, or if `irow` is not in `[0,n1-1]`, or if `jcol` is not in `[0,n2-1]`, an error message is printed and the program exits.
    ```

1.2.3 Initialize methods

1. `void A2_init ( A2 *mtx, int type, int n1, int n2, int inc1, int inc2, double *entries ) ;`

   This is the basic initializer method. We require that `mtx` not be `NULL`, `type` be either `SPOOLES_REAL` or `SPOOLES_COMPLEX`, `n1` and `n2` both be positive, and both `inc1` and `inc2` both be positive and that one of them be equal to one. Also, we only initialize a full matrix, i.e., one of `inc1 = 1` and `inc2 = nrow` or `inc1 = ncol` and `inc2 = 1` must hold.

   The object is first cleared with a call to `A2_clearData()`. If `entries` is `NULL` then `n1*n2` new entries are found, `mtx->entries` is set to this address and `nowned` is set to `n1*n2`. If `entries` is not `NULL`, then `mtx->entries` is set to `entries` and `nowned` is set to zero.

   Error checking: If `mtx` is `NULL`, or if `n1`, `n2`, `inc1` or `inc2` are less than or equal to zero, or if the matrix is not full matrix (i.e., `inc1` must be `1` and `inc2` must be `n1`, *or* `inc1` must be `n2` and `inc2` must be `1`), an error message is printed and zero is returned.

2. `void A2_subA2 ( A2 *mtxA, A2 *mtxB, int firstrow, int lastrow, int firstcol, int lastcol ) ;`

   This initializer method makes the object `mtxA` point into a submatrix of object `mtxB`, as

   ```
   A(0:lastrow-firstrow,0:lastcol-firstcol) = B(firstrow:lastrow, firstcol:lastcol)
   ```

   Note, `firstrow`, `lastrow`, `firstcol` and `lastcol` must satisfy `0 <= firstrow <= lastrow < mtxB->n1` and `0 <= firstcol <= lastcol < mtxB->n2`. Object `mtxA` does not own its entries, but points into the entries of `mtxB`.

   Error checking: If `mtxA` or `mtxB` are `NULL`, or if `firstrow` or `lastrow` are out of range, or if `firstcol` or `lastcol` are out of range, an error message is printed and zero is returned.

1.2.4 Methods used in the QR factorization

1. `void A2_makeStaircase ( A2 *mtxA ) ;`

   This method permutes the rows of `mtxA` by the location of the leading nonzero of each row. Upon return, the matrix is in *staircase* form.

   Error checking: If `mtxA` is `NULL`, an error message is printed and the program exits.

2. `double A2_QRreduce ( A2 *mtxA, DV *workDV, int msglvl, FILE *msgFile ) ;`

   This method computes $A = QR$ factorization. On return, the matrix Q is not available, and R is found in the upper triangle or upper trapezoid of `mtxA`. The Householder vectors are stored in the lower triangle of `mtxA`, with $v_j(j) = 1.0$. The return value is the number of floating point operations that were executed.

   Error checking: If `mtxA` or `workDV` is `NULL`, or if `msglvl > 0` and `msgFile` if `NULL`, an error message is printed and the program exits.

3. `void A2_computeQ ( A2 *mtxQ, A2 *mtxA, DV *workDV, int msglvl, FILE *msgFile ) ;`

   This method computes Q from the $A = QR$ factorization computed in `A2_QRreduce()`. Note: `mtxA` and `mtxQ` must be column major.

   Error checking: If `mtxQ`, `mtxA` or `workDV` is `NULL`, or if `msglvl > 0` and `msgFile` if `NULL`, an error message is printed and the program exits.

1.2.5 Norm methods

These methods return a norm of a row or a column, or the easily computable norms of the matrix.

1. `double A2_maxabs ( A2 *mtx ) ;`

   This method returns magnitude of the entry with largest magnitude.

   Error checking: If `mtx` is `NULL`, an error message is printed and the program exits.

2. `double A2_frobNorm ( A2 *mtx ) ;`

   This method returns the Frobenius norm of the matrix.

   Error checking: If `mtx` is `NULL`, an error message is printed and the program exits.

3. `double A2_oneNorm ( A2 *mtx ) ;`

   This method returns the one norm of the matrix.

   Error checking: If `mtx` is `NULL`, an error message is printed and the program exits.

4. `double A2_infinityNorm ( A2 *mtx ) ;`

   This method returns the infinity norm of the matrix.

   Error checking: If `mtx` is `NULL`, an error message is printed and the program exits.

5. `double A2_oneNormOfColumn ( A2 *mtx, int jcol ) ;`

   This method returns the one-norm of column `jcol` of the matrix.

   Error checking: If `mtx` is `NULL`, or `jcol` is not in `[0,n2-1]`, an error message is printed and the program exits.

6. `double A2_twoNormOfColumn ( A2 *mtx, int jcol ) ;`

   This method returns the two-norm of column `jcol` of the matrix.

   Error checking: If `mtx` is `NULL`, or `jcol` is not in `[0,n2-1]`, an error message is printed and the program exits.

7. `double A2_infinityNormOfColumn ( A2 *mtx, int jcol ) ;`

   This method returns the infinity-norm of column `jcol` of the matrix.

   Error checking: If `mtx` is `NULL`, or `jcol` is not in `[0,n2-1]`, an error message is printed and the program exits.

8. `double A2_oneNormOfRow ( A2 *mtx, int irow ) ;`

   This method returns the one-norm of row `irow` of the matrix.

   Error checking: If `mtx` is `NULL`, or `irow` is not in `[0,n1-1]`, an error message is printed and the program exits.

9. `double A2_twoNormOfRow ( A2 *mtx, int irow ) ;`

   This method returns the two-norm of row `irow` of the matrix.

   Error checking: If `mtx` is `NULL`, or `irow` is not in `[0,n1-1]`, an error message is printed and the program exits.

10. `double A2_infinityNormOfRow ( A2 *mtx, int irow ) ;`

    This method returns the infinity-norm of row `irow` of the matrix.

    Error checking: If `mtx` is `NULL`, or `irow` is not in `[0,n1-1]`, an error message is printed and the program exits.

1.2.6 Sort methods

1. `void A2_permuteRows ( A2 *mtx, int nrow, int index[] ) ;`

   The `index[]` vector contains the *row ids* of the leading `nrow` rows. This method permutes the leading `nrow` rows of the matrix so that the `index[]` vector is in ascending order. This method calls `A2_permuteRows()` but does not overwrite the `index[]` vector.

   Error checking: If `mtx` or `index[]` is `NULL`, or if `nrow < 0` or `nrow > n1`, an error message is printed and the program exits.

2. `void A2_permuteColumns ( A2 *mtx, int nrow, int index[] ) ;`

   The `index[]` vector contains the *column ids* of the leading `ncol` rows. This method permutes the leading `ncol` columns of the matrix so that the `index[]` vector is in ascending order. This method calls `A2_permuteColumns()` but does not overwrite the `index[]` vector.

   Error checking: If `mtx` or `index[]` is `NULL`, or if `ncol < 0` or `ncol > n2`, an error message is printed and the program exits.

3. `int A2_sortRowsUp ( A2 *mtx, int nrow, int rowids[] ) ;`

   This method sorts the leading `nrow` rows of the matrix into ascending order with respect to the `rowids[]` vector. The return value is the number of row swaps made.

   Error checking: If `mtx` or `rowids` is `NULL`, or if `nrow < 0` or `nrow > n1`, an error message is printed and the program exits.

4. `int A2_sortColumnsUp ( A2 *mtx, int ncol, int colids[] ) ;`

   This method sorts the leading `ncol` columnss of the matrix into ascending order with respect to the `colids[]` vector. The return value is the number of column swaps made.

   Error checking: If `mtx` or `colids` is `NULL`, or if `ncol < 0` or `ncol > n2`, an error message is printed and the program exits.

1.2.7 Utility methods

1. `int A2_sizeOf ( A2 *mtx ) ;`

   This method returns the number of bytes owned by this object.

   Error checking: If `mtx` is `NULL` an error message is printed and the program exits.

2. `void A2_shiftBase ( A2 *mtx, int rowoff, int coloff ) ;`

   This method is used to shift the base of the entries and adjust dimensions of the `A2` object.

   ```
   mtx(0:n1-rowoff-1,0:n2-coloff-1) := mtx(rowoff:n1-1,coloff:n2-1)
   ```

   Error checking: If `mtx` is `NULL` an error message is printed and the program exits.

3. `int A2_rowMajor ( A2 *mtx ) ;`

   This method returns 1 if the storage is row major, otherwise it returns zero.

   Error checking: If `mtx` is `NULL`, an error message is printed and the program exits.

4. `int A2_columnMajor ( A2 *mtx ) ;`

   This method returns 1 if the storage is column major, otherwise it returns zero.

   Error checking: If `mtx` is `NULL`, an error message is printed and the program exits.

5. `void A2_transpose ( A2 *mtx ) ;`

   This method replaces `mtx` with its transpose. Note, this takes $O(1)$ operations since we just swap dimensions and increments.

   Error checking: If `mtx` is `NULL`, an error message is printed and the program exits.

6. `void A2_extractRow ( A2 *mtx, double row[], int irow ) ;`

   This method fills the `row[]` vector with row `irow` of the matrix.

   Error checking: If `mtx`, `entries` or `row` are `NULL`, or if `irow` is not in `[0,n1-1]`, an error message is printed and the program exits.

7. `void A2_extractRowDV ( A2 *mtx, DV *rowDV, int irow ) ;`

   This method fills the `rowDV` object with row `irow` of the matrix.

   Error checking: If `mtx` or `rowDV` are `NULL`, or if the matrix is not real, or if `irow` is not in `[0,n1-1]`, an error message is printed and the program exits.

8. `void A2_extractRowZV ( A2 *mtx, ZV *rowZV, int irow ) ;`

   This method fills the `rowZV` object with row `irow` of the matrix.

   Error checking: If `mtx` or `rowZV` are `NULL`, or if the matrix is not complex, or if `irow` is not in `[0,n1-1]`, an error message is printed and the program exits.

9. `void A2_extractColumn ( A2 *mtx, double col[], int jcol ) ;`

   This method fills the `col[]` vector with column `jcol` of the matrix.

   Error checking: If `mtx`, `entries` or `col` are `NULL`, or if `jcol` is not in `[0,n2-1]`, an error message is printed and the program exits.

10. `void A2_extractColumnDV ( A2 *mtx, DV *colDV, int jcol ) ;`

    This method fills the `colDV` object with column `jcol` of the matrix.

    Error checking: If `mtx` or `colDV` are `NULL`, or if the matrix is not complex, or if `jcol` is not in `[0,n2-1]`, an error message is printed and the program exits.

11. `void A2_extractColumnZV ( A2 *mtx, ZV *colZV, int jcol ) ;`

    This method fills the `colZV` object with column `jcol` of the matrix.

    Error checking: If `mtx` or `colZV` are `NULL`, or if the matrix is not complex, or if `jcol` is not in `[0,n2-1]`, an error message is printed and the program exits.

12. `void A2_setRow ( A2 *mtx, double row[], int irow ) ;`

    This method fills row `irow` of the matrix with the entries in the `row[]` vector.

    Error checking: If `mtx`, `entries` or `row[]` are `NULL`, or if `irow` is not in `[0,n1-1]`, an error message is printed and the program exits.

13. `void A2_setRowDV ( A2 *mtx, DV rowDV, int irow ) ;`

    This method fills row `irow` of the matrix with the entries in the `rowDV` object.

    Error checking: If `mtx` or `rowDV` are `NULL`, or if the matrix is not real, or if `irow` is not in `[0,n1-1]`, an error message is printed and the program exits.

14. `void A2_setRowZV ( A2 *mtx, ZV rowZV, int irow ) ;`

    This method fills row `irow` of the matrix with the entries in the `rowZV` object.

    Error checking: If `mtx` or `rowZV` are `NULL`, or if the matrix is not complex, or if `irow` is not in `[0,n1-1]`, an error message is printed and the program exits.

15. `void A2_setColumn ( A2 *mtx, double col[], int jcol ) ;`

    This method fills column `jcol` of the matrix with the entries in the `col[]` vector.

    Error checking: If `mtx` or `colZV` are `NULL`, or if `jcol` is not in `[0,n2-1]`, an error message is printed and the program exits.

16. `void A2_setColumnDV ( A2 *mtx, DV colDV, int jcol ) ;`

    This method fills column `jcol` of the matrix with the entries in the `colDV` object.

    Error checking: If `mtx` or `colDV` are `NULL`, or if the matrix is not complex, or if `jcol` is not in `[0,n2-1]`, an error message is printed and the program exits.

17. `void A2_setColumnZV ( A2 *mtx, ZV colZV, int jcol ) ;`

    This method fills column `jcol` of the matrix with the entries in the `colZV` object.

    Error checking: If `mtx` or `colZV` are `NULL`, or if the matrix is not complex, or if `jcol` is not in `[0,n2-1]`, an error message is printed and the program exits.

18. `void A2_fillRandomUniform ( A2 *mtx, double lower, double upper, int seed ) ;`

    This method fills the matrix with random numbers taken from a uniform distribution on `[lower,upper]` using the `Drand` object.

    Error checking: If `mtx` is `NULL`, an error message is printed and the program exits.

19. `void A2_fillWithIdentity ( A2 *mtx ) ;`

    This method fills the matrix with the identity matrix.

    Error checking: If `mtx` is `NULL` or if `n1 != n2`, an error message is printed and the program exits.

20. `void A2_zero ( A2 *mtx ) ;`

    This method fills the matrix with zeros.

    Error checking: If `mtx` is `NULL`, an error message is printed and the program exits.

21. `void A2_copy ( A2 *mtxA, A2 *mtxB ) ;`

    This method copies entries from matrix `mtxB` into matrix `mtxA`. Note, `mtxA` and `mtxB` need not be of the same size, the leading `min(mtxA->n1,mtxB->n1)` rows and `min(mtxA->n2,mtxB->n2)` columns are copied.

    Error checking: If `mtxA` or `mtxB` is `NULL`, or if the matrices are not of the same type, an error message is printed and the program exits.

22. `void A2_sub ( A2 *mtxA, A2 *mtxB ) ;`

    This method subtracts entries in matrix `mtxB` from entries in matrix `mtxA`. Note, `mtxA` and `mtxB` need not be of the same size, the leading `min(mtxA->n1,mtxB->n1)` rows and `min(mtxA->n2,mtxB->n2)` columns are subtracted.

    Error checking: If `mtxA` or `mtxB` is `NULL`, or if the matrices are not of the same type, an error message is printed and the program exits.

23. `void A2_swapRows ( A2 *mtx, int irow1, int irow2 ) ;`

    This method swaps rows `irow1` and `irow2` of the matrix.

    Error checking: If `mtxA` or `mtxB` is `NULL`, or if `irow1` or `irow2` are out of range, an error message is printed and the program exits.

24. `void A2_swapColumns ( A2 *mtx, int irow1, int irow2 ) ;`

    This method swaps columns `jcol1` and `jcol2` of the matrix.

    Error checking: If `mtxA` or `mtxB` is `NULL`, or if `jcol1` or `jcol1` are out of range, an error message is printed and the program exits.

25. ```
    int A2_copyEntriesToVector ( A2 *mtx, int length, double dvec[],
                                 int copyflag, int storeflag ) ;
    ```

    This method copies selected entries from `mtx` into the vector `dvec[]` with length `length`. The return value is the number of entries copied. This method is used during the QR factorization to extract factor entries and update matrix entries from a front. All entries may be copied, or only the diagonal, lower or upper entries, and the entries may be copied to `dvec[]` by rows or by columns.

    Error checking: If `mtx` or `dvec` is `NULL`, or if `length` is not as large as the number of entries to be copied, or if `copyflag` is not one of `A2_STRICT_LOWER`, `A2_LOWER`, `A2_DIAGONAL`, `A2_UPPER`, `A2_STRICT_UPPER` or `A2_ALL_ENTRIES`, or if `storeflag` is not one of `A2_BY_ROWS` or `A2_BY_COLUMNS`, an error message is printed and the program exits.

1.2.8 IO methods

There are the usual eight IO routines plus a method to write the object to a Matlab file.

1. `int A2_readFromFile ( A2 *mtx, char *fn ) ;`

   This method reads a `A2` object from a file. It tries to open the file and if it is successful, it then calls `A2_readFromFormattedFile()` or `A2_readFromBinaryFile()`, closes the file and returns the value returned from the called routine.

   Error checking: If `mtx` or `fn` are `NULL`, or if `fn` is not of the form `*.a2f` (for a formatted file) or `*.a2b` (for a binary file), an error message is printed and the method returns zero.

2. `int A2_readFromFormattedFile ( A2 *mtx, FILE *fp ) ;`

   This method reads a `A2` object from a formatted file whose pointer is `fp`. If there are no errors in reading the data, the value `1` is returned. If an IO error is encountered from `fscanf`, zero is returned.

   Error checking: If `mtx` or `fp` are `NULL`, an error message is printed and zero is returned.

3. `int A2_readFromBinaryFile ( A2 *mtx, FILE *fp ) ;`

   This method reads a `A2` object from a binary file whose pointer is `fp`. If there are no errors in reading the data, the value `1` is returned. If an IO error is encountered from `fread`, zero is returned.

   Error checking: If `mtx` or `fp` are `NULL`, an error message is printed and zero is returned.

4. `int A2_writeToFile ( A2 *mtx, char *fn ) ;`

   This method writes a `A2` object to a file. It tries to open the file and if it is successful, it then calls `A2_writeFromFormattedFile()` or `A2_writeFromBinaryFile()`, closes the file and returns the value returned from the called routine.

   Error checking: If `mtx` or `fn` are `NULL`, or if `fn` is not of the form `*.a2f` (for a formatted file) or `*.a2b` (for a binary file), an error message is printed and the method returns zero.

5. `int A2_writeToFormattedFile ( A2 *mtx, FILE *fp ) ;`

   This method writes a `A2` object to a formatted file whose pointer is `fp`. If there are no errors in writing the data, the value `1` is returned. If an IO error is encountered from `fprintf`, zero is returned.

   Error checking: If `mtx` or `fp` are `NULL`, an error message is printed and zero is returned.
    ```

6. `int A2_writeToBinaryFile ( A2 *mtx, FILE *fp ) ;`

   This method writes a `A2` object to a binary file whose pointer is `fp`. If there are no errors in writing the data, the value `1` is returned. If an IO error is encountered from `fwrite`, zero is returned.

   Error checking: If `mtx` or `fp` are `NULL`, an error message is printed and zero is returned.

7. `void A2_writeForHumanEye ( A2 *mtx, FILE *fp ) ;`

   This method writes a `A2` object to a file in an easily readable format. The method `A2_writeStats()` is called to write out the header and statistics.

   Error checking: If `mtx` or `fp` are `NULL`, an error message is printed and zero is returned.

8. `void A2_writeStats ( A2 *mtx, FILE *fp ) ;`

   This method writes a header and some statistics to a file.

   Error checking: If `mtx` or `fp` are `NULL`, an error message is printed and zero is returned.

9. `void A2_writeForMatlab ( A2 *mtx, char *mtxname, FILE *fp ) ;`

   This method writes the entries of the matrix to a file in Matlab format. The name of the matrix is `mtxname`.

   Error checking: If `mtx`, `mtxname` or `fp` are `NULL`, an error message is printed and zero is returned.

1.3 Driver programs for the `A2 object`

1. `test_norms msglvl msgFile type nrow ncol inc1 inc2 seed`

   This driver program tests the `A2` norm methods. Use the script file `do_norms` for testing. When the output file is loaded into matlab, the last two lines contain matrices whose entries should all be around machine epsilon.

   - The `msglvl` parameter determines the amount of output. Use `msglvl = 1` for just timing output.
   - The `msgFile` parameter determines the message file — if `msgFile` is `stdout`, then the message file is *stdout*, otherwise a file is opened with *append* status to receive any output data.
   - The `type` parameter denotes the type of entries — `SPOOLES_REAL` or `SPOOLES_COMPLEX`
   - The `nrow` parameter is the number of rows.
   - The `ncol` parameter is the number of rows.
   - The `inc1` parameter is the row increment.
   - The `inc2` parameter is the column increment.
   - The `seed` parameter is a random number seed.

2. `test_QR msglvl msgFile type nrow ncol inc1 inc2 seed`

   This driver program tests the `A2_QRreduce()` and `A2_QRreduce2()` methods which reduce A to QR via rank-1 and rank-2 updates. Use the script file `do_QR` for testing. When `msglvl > 1`, the matrix A and matrices $R1$ and $R2$ (computed from `A2_QRreduce()` and `A2_QRreduce2()`, respectively) are printed to the message file. When the output file is loaded into matlab, the errors $A^TA - R_1^TR_1$ and $A^TA - R_2^TR_2$ (if A is real) or the errors $A^HA - R_1^HR_1$ and $A^HA - R_2^HR_2$ (if A is complex) are computed.

   - The `msglvl` parameter determines the amount of output. Use `msglvl = 1` for just timing output.
   - The `msgFile` parameter determines the message file — if `msgFile` is `stdout`, then the message file is *stdout*, otherwise a file is opened with *append* status to receive any output data.

- The `type` parameter denotes the type of entries — `SPOOLES_REAL` or `SPOOLES_COMPLEX`
- The `nrow` parameter is the number of rows.
- The `ncol` parameter is the number of rows.
- The `inc1` parameter is the row increment.
- The `inc2` parameter is the column increment.
- The `seed` parameter is a random number seed.

Index