Article

Creating sparse matrices

Create sparse matrices for factorization and solving systems.

Overview

In the Accelerate framework, the Sparse Solvers library stores sparse matrices using the compressed sparse column (CSC) format. CSC stores a matrix as a series of column vectors that specify the nonzero entries as `(row-index, value)` pairs and omit the zero entries.

The Sparse Solvers library provides routines to convert matrices from other formats to CSC. For more information, see Conversion from Other Formats.

The Sparse Solvers library supports unsymmetric and symmetric sparse matrices, each of which can also be block matrices.

- An *unsymmetric matrix* contains either `Double` or `Float` values with no symmetry between its lower-left and upper-right triangles.
- A *symmetric matrix* is symmetrical along the diagonal from its upper-left to lower-right corners. In other words, a symmetric matrix is equal to its transpose ($A=A^T$).
- A *block matrix* can be either unsymmetric or symmetric, and consists of sections called blocks. The blocks along the diagonal of a symmetric block matrix must, themselves, be symmetrical.

Create an unsymmetric matrix

In this example of an unsymmetric sparse matrix, empty cells represent zeros:

$$\begin{bmatrix} 2.0 & 1.0 & \\ -0.2 & 3.2 & 1.4 \\ & -0.1 & 0.5 \\ 2.5 & 1.1 & \end{bmatrix}$$

The first step to create a matrix is to define two arrays that store the row indices and corresponding values.

Swift Objective-C

```
var rowIndices: [Int32] = [0, 1, 3,      // Column 0
                           0, 1, 2, 3,   // Column 1
                           1, 2]         // Column 2

var values = [2.0, -0.2, 2.5,            // Column 0
              1.0, 3.2, -0.1, 1.1,       // Column 1
              1.4, 0.5]                  // Column 2
```

In addition to the `(row-index, value)` pairs, create a third array that specifies where each column starts. This array requires an additional, final entry that defines the final column's length.

In the following example, the zeroth item in the `values` array starts column 0, the third starts column 1, and the seventh starts column 2:

Swift Objective-C

```
var columnStarts = [0,     // Column 0
                    3,     // Column 1
                    7,     // Column 2
                    9]
```

The two structural arrays, `rowIndices` and `columnStarts`, create a `SparseMatrixStructure` instance that describes the matrix's structure. The initializer requires an attributes object, and the default parameters of a `SparseAttributes_t` instance specify an unsymmetric matrix.

Swift Objective-C

```
let structure = SparseMatrixStructure(rowCount: 4,
                                      columnCount: 3,
                                      columnStarts: &columnStarts,
                                      rowIndices: &rowIndices,
                                      attributes: SparseAttributes_t(),
                                      blockSize: 1)
```

The following code uses the `structure` and `values` items to create a `SparseMatrix_Double` instance:

Swift Objective-C

```swift
values.withUnsafeMutableBufferPointer { valuesPtr in
    let A = SparseMatrix_Double(
        structure: structure,
        data: valuesPtr.baseAddress!
    )

    // Perform operations using `A`.
}
```

Create a symmetric matrix

In this example of a symmetric sparse matrix, empty cells represent zeros:

$$\begin{bmatrix} 10.0 & 1.0 & & 2.5 \\ 1.0 & 12.0 & -0.3 & 1.1 \\ & -0.3 & 9.5 & \\ 2.5 & 1.1 & & 6.0 \end{bmatrix}$$

Because it’s symmetric, the values in the upper triangle of the matrix are redundant, so exclude them from the data that you pass to the `SparseMatrix_Double` initializer. The example below shows the excluded values in gray:

$$\begin{bmatrix} 10.0 & 1.0 & & 2.5 \\ 1.0 & 12.0 & -0.3 & 1.1 \\ & -0.3 & 9.5 & \\ 2.5 & 1.1 & & 6.0 \end{bmatrix}$$

As with the unsymmetric example, the `rowIndices` array specifies the row in the matrix that contains the corresponding item in `values`, and the `columnStarts` array specifies where each column starts in the `rowIndices` array.

In the following example, the `attributes` parameter specifies that the matrix is symmetric and the items in the values array derive from the lower triangle:

Swift Objective-C

```
var columnStarts = [0,                  // Column 0
                    3,                  // Column 1
                    6,                  // Column 2
                    7,                  // Column 3
                    8]

var rowIndices: [Int32] = [0, 1, 3,     // Column 0
                           1, 2, 3,     // Column 1
                           2,           // Column 2
                           3]           // Column 3

var attributes = SparseAttributes_t()
attributes.triangle = SparseLowerTriangle
attributes.kind = SparseSymmetric

let structure = SparseMatrixStructure(rowCount: 4,
                                      columnCount: 4,
                                      columnStarts: &columnStarts,
                                      rowIndices: &rowIndices,
                                      attributes: attributes,
                                      blockSize: 1)
```

Create the SparseMatrix_Double instance using the structure from the code example above and the values from the lower triangle of the matrix.

Swift   Objective-C

```
var values = [10.0,  1.0, 2.5,  // Column 0
              12.0, -0.3, 1.1,  // Column 1
              9.5,              // Column 2
              6.0]              // Column 3

values.withUnsafeMutableBufferPointer { valuesPtr in
    let A = SparseMatrix_Double(
        structure: structure,
        data: valuesPtr.baseAddress!
    )

    // Perform operations using `A`.
}
```

Create a block matrix

You can create block sparse matrices — that is, a matrix that consists of blocks that contain multiple values — by defining a `blockSize` greater than 1. The block size is the length of the side of the square block.

Block matrices can be symmetric or unsymmetric. This example shows an unsymmetric sparse matrix with a block size of 3:

$$\left[\begin{array}{ccc|ccc|ccc}
1.0 & 0.3 & 0.2 & 1.5 & 0.2 & 0.3 & & & \\
0.1 & 0.5 & 1.3 & 2.5 & 0.8 & 0.4 & & & \\
9.2 & 1.3 & 4.5 & 7.2 & 0.8 & 0.2 & & & \\
\hline
 & & & 0.2 & 1.6 & 0.5 & & & \\
 & & & 0.4 & 1.8 & 4.2 & & & \\
 & & & 1.8 & 0.6 & 3.3 & & & \\
\hline
0.2 & 1.8 & 0.8 & & & & 0.2 & 0.4 & 1.8 \\
0.7 & 1.6 & 0.7 & & & & 0.8 & 0.6 & 1.2 \\
0.9 & 1.7 & 0.9 & & & & 1.2 & 0.8 & 0.9
\end{array}\right]$$

The following example shows the code to create a sparse matrix with the structure and values above. The `SparseMatrixStructure` specifies a block size of 3. The values for each block concatenate in column-major order.

Swift   Objective-C

```swift
var columnStarts = [ 0, 2, 4, 5 ]
var rowIndices: [Int32] = [ 0, 2, 0, 1, 2 ]

var values = [
    1.0, 0.1, 9.2, 0.3, 0.5, 1.3, 0.2, 1.3, 4.5,    // Block 0
    0.2, 0.7, 0.9, 1.8, 1.6, 1.7, 0.8, 0.7, 0.9,    // Block 1
    1.5, 2.5, 7.2, 0.2, 0.8, 0.8, 0.3, 0.4, 0.2,    // Block 2
    0.2, 0.4, 1.8, 1.6, 1.8, 0.6, 0.5, 4.2, 3.3,    // Block 3
    0.2, 0.8, 1.2, 0.4, 0.6, 0.8, 1.8, 1.2, 0.9     // Block 4
]
var attributes = SparseAttributes_t()
attributes.kind = SparseOrdinary

let structure = SparseMatrixStructure(rowCount: 3,
                                      columnCount: 3,
                                      columnStarts: &columnStarts,
                                      rowIndices: &rowIndices,
                                      attributes: attributes,
                                      blockSize: 3)
```

```
values.withUnsafeMutableBufferPointer { valuesPtr in
    let A = SparseMatrix_Double(
        structure: structure,
        data: valuesPtr.baseAddress!
    )

    // Perform operations using `A`.
}
```

When you create a symmetric matrix with a block size greater than 1, the blocks along the matrix's diagonal must also be symmetric.

See Also

Sparse Matrices

Solving systems using direct methods

Use direct methods to solve systems of equations where the coefficient matrix is sparse.

Solving systems using iterative methods

Use iterative methods to solve systems of equations where the coefficient matrix is sparse.

Creating a sparse matrix from coordinate format arrays

Use separate coordinate format arrays to create sparse matrices.

Sparse Solvers

Solve systems of equations where the coefficient matrix is sparse.