Article

Solving systems using direct methods

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

Overview

Direct methods offer high-precision solving with a simple API when compared to iterative methods. The code example below uses sparse Cholesky factorization to solve the following equation:

$$\begin{bmatrix} 10.0 & 1.0 & & 2.5 \\ 1.0 & 12.0 & -0.3 & 1.1 \\ & -0.3 & 9.5 & \\ 2.5 & 1.1 & & 6.0 \end{bmatrix} x = \begin{bmatrix} 2.2 \\ 2.85 \\ 2.79 \\ 2.87 \end{bmatrix}$$

In the equation above, *A* refers to the four-by-four coefficient matrix and *b* to the right-hand-side vector. The code in this article solves the equation *Ax = b* by finding *x*.

Note that *A* is sparse. Some entries (those that are blank) are zero. For small matrices such as this, there's little gain in exploiting this structure. However, it's essential for larger systems that don't otherwise fit in memory, even on a large computer.

The code in this article performs a sparse Cholesky factorization, equivalent to calling the LAPACK function `dpotrf_(_:_:_:_:_:)` on a dense matrix. The main requirement for sparse Cholesky factorization is that the matrix is symmetric positive-definite (that is, $A=A^T$), and all eigenvalues are greater than zero. A sufficient, but not necessary, condition is that the matrix is diagonally dominant (that is, the sum of the absolute values of the off-diagonal entries in each row or column is less than the value of the diagonal). This is the case for the above matrix.

Create the matrix structure

Use the code below to define the matrix structure. As Creating sparse matrices explains, because *A* is symmetric, it stores only half of the data.

Swift Objective-C

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

var columnStarts = [0, 3, 6, 7, 8]

let structure: SparseMatrixStructure = rowIndices.withUnsafeMutableBufferPointer { r
    columnStarts.withUnsafeMutableBufferPointer { columnStartsPtr in
        var attributes = SparseAttributes_t()
        attributes.triangle = SparseLowerTriangle
        attributes.kind = SparseSymmetric

        return SparseMatrixStructure(
            rowCount: 4,
            columnCount: 4,
            columnStarts: columnStartsPtr.baseAddress!,
            rowIndices: rowIndicesPtr.baseAddress!,
            attributes: attributes,
            blockSize: 1
        )
    }
}
```

Create and factorize the matrix

The `SparseFactor(_:_:)` function performs the actual Cholesky factorization, finding *L* such that $A = LL^T$.

Swift Objective-C

```
let llt: SparseOpaqueFactorization_Double = values.withUnsafeMutableBufferPointer {
    let a = SparseMatrix_Double(
        structure: structure,
        data: valuesPtr.baseAddress!
    )

    return SparseFactor(SparseFactorizationCholesky, a)
}
```

If the factorization function encounters an error, the code prints an error message and terminates. You may instead want to capture the error by using the optional `SparseSymbolicFactorOptions` parameter and set the `reportError` parameter to a user-supplied error-handling routine. The returned `SparseOpaqueFactorization_Double` structure reflects the error.

Solve the equation

Use the factorization to solve the equation. The right-hand-side and solution vectors are arrays that you wrap in `DenseVector_Double` structures. The actual values of `x` don't matter because the function overwrites them.

Swift   Objective-C

```
var bValues = [ 2.20, 2.85, 2.79, 2.87 ]
var xValues = [ 0.00, 0.00, 0.00, 0.00 ]
```

The solve call takes the factorization $A = LL^T$ and solves the system $Ax = b$ as $LL^Tx = b$ by solving the two triangular systems:

- $Ly = b$
- $L^Tx = y$

However, you need only to supply the right-hand-side vector and the factorization.

The `SparseSolve(_:_:_:)` function solves the equation and populates `x` with the solution.

Swift   Objective-C

```
bValues.withUnsafeMutableBufferPointer { bPtr in
    xValues.withUnsafeMutableBufferPointer { xPtr in

        let b = DenseVector_Double(
            count: 4,
            data: bPtr.baseAddress!
        )
        let x = DenseVector_Double(
            count: 4,
            data: xPtr.baseAddress!
        )

        SparseSolve(llt, b, x)
    }
```

If the `SparseSolve(_:_:_:)` function encounters an error, the code prints an error message and terminates, unless you set `reportError` on the initial call to `SparseFactor(_:_:)`.

The following code iterates over the solution vector, `x`, and prints the solution, `x = 0.10 0.20 0.30 0.40`.

Swift   Objective-C

```
print("x = \(xValues)")
```

See Also

Sparse Matrices

Creating sparse matrices

Create sparse matrices for factorization and solving systems.

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.