Article

Solving systems using iterative methods

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

Overview

The code in this article solves the following equation by using the iterative method of least squares minimum residual (LSMR) to find the solution.

$$\begin{bmatrix} 2.0 & 1.0 & \\ -0.2 & 3.2 & 1.4 \\ & -0.1 & 0.5 \\ 2.5 & 1.1 & \end{bmatrix} x = \begin{bmatrix} 1.200 \\ 1.013 \\ 0.205 \\ -0.172 \end{bmatrix}$$

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

Because `A` is an overdetermined matrix (that is, it has more rows than columns), for most right-hand-sides there isn't an exact solution. In this case, you usually find the closest solution that minimizes the 2-norm of the error. That is, the solution solves the optimization $\min \| Ax - b \|_2$. This is known as the *least-squares problem*.

You could solve this problem through a direct method, such as sparse QR; however, for some problems, a faster method that provides an adequate solution is the iterative method LSMR. Unlike direct methods that factorize the matrix `A`, iterative methods only require the ability to multiply by the matrix (and its transpose, A^T). They move through a sequence of approximate solutions, converging to the correct answer. However, these methods run into numerical difficulties more often than direct methods. Resolving these issues requires expert knowledge, and is sometimes impossible. The most common method to improve and accelerate convergence is to use a preconditioner — an operator that approximates A^{-1}. For least-squares problems, using a diagonal matrix with entries equal to the 2-norm of each column is often sufficient and is the method that this article covers.

Create the matrix

Use the code below — which Creating sparse matrices covers in detail — to define the unsymmetric matrix A:

Swift Objective-C

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

var columnStarts = [0, 3, 7, 9]

let structure: SparseMatrixStructure = rowIndices.withUnsafeMutableBufferPointer { r
    columnStarts.withUnsafeMutableBufferPointer { columnStartsPtr in
        let attributes = SparseAttributes_t()

        return SparseMatrixStructure(
            rowCount: 4,
            columnCount: 3,
            columnStarts: columnStartsPtr.baseAddress!,
            rowIndices: rowIndicesPtr.baseAddress!,
            attributes: attributes,
            blockSize: 1
        )
    }
}

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

var xValues = [ 0.0, 0.0, 0.0 ]
var bValues = [1.200, 1.013, 0.205, -0.172]

values.withUnsafeMutableBufferPointer { valuesPtr in

    let a = SparseMatrix_Double(
        structure: structure,
        data: valuesPtr.baseAddress!
    )
```

```
[...]
```

Solve the equation

Define the right-hand-side and solution vectors as arrays that you wrap in `DenseVector_Double` structures. The sparse solve function uses the initial values of `x` as an initial guess of the solution. If you don't have a good estimate, initialize all the values to zero.

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: 3,
                data: xPtr.baseAddress!
            )

[...]
```

Use the matrix and vectors to perform the full LSMR iteration and iterate over the results in `x`. `SparseSolve(_:_:_:_:)` returns a status which, if equal to `SparseIterativeConverged`, indicates that the vector, `x`, contains the solution.

Swift | Objective-C

```
[...]
            let status = SparseSolve(SparseLSMR(), a, b, x, SparsePreconditionerDiag
            if status != SparseIterativeConverged {
                fatalError("Failed to converge. Returned with error \(status).")
            }
        }
    }
}

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

On return, x contains the values `0.10 0.20 0.30`.

See Also

Sparse Matrices

- Creating sparse matrices

  Create sparse matrices for factorization and solving systems.

- Solving systems using direct methods

  Use direct 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.