
Sparse Linear Algebra

This chapter describes functions for solving sparse linear systems of equations. The library provides linear algebra routines which operate directly on the `gsl_spmatrix` and `gsl_vector` objects.

The functions described in this chapter are declared in the header file `gsl_splinalg.h`.

Overview

This chapter is primarily concerned with the solution of the linear system

$$Ax = b$$

where A is a general square n-by-n non-singular sparse matrix, x is an unknown n-by-1 vector, and b is a given n-by-1 right hand side vector. There exist many methods for solving such sparse linear systems, which broadly fall into either direct or iterative categories. Direct methods include LU and QR decompositions, while iterative methods start with an initial guess for the vector x and update the guess through iteration until convergence. GSL does not currently provide any direct sparse solvers.

Sparse Iterative Solvers

Overview

Many practical iterative methods of solving large n-by-n sparse linear systems involve projecting an approximate solution for `x` onto a subspace of $\mathbf{R}^n$. If we define a m-dimensional subspace $\mathcal{K}$ as the subspace of approximations to the solution `x`, then m constraints must be imposed to determine the next approximation. These m constraints define another m-dimensional subspace denoted by $\mathcal{L}$. The subspace dimension m is typically chosen to be much smaller than n in order to reduce the computational effort needed to generate the next approximate solution vector. The many iterative algorithms which exist differ mainly in their choice of $\mathcal{K}$ and $\mathcal{L}$.

Types of Sparse Iterative Solvers

The sparse linear algebra library provides the following types of iterative solvers:

`gsl_splinalg_itersolve_type`

`gsl_splinalg_itersolve_gmres`

This specifies the Generalized Minimum Residual Method (GMRES). This is a projection method using $\mathcal{K} = \mathcal{K}_m$ and $\mathcal{L} = A\mathcal{K}_m$ where $\mathcal{K}_m$ is the m-th Krylov subspace

$$\mathcal{K}_m = span \left\{ r_0, Ar_0, ..., A^{m-1} r_0 \right\}$$

and $r_0 = b - Ax_0$ is the residual vector of the initial guess x_0. If m is set equal to n, then the Krylov subspace is $\mathbf{R}^n$ and GMRES will provide the exact solution `x`. However, the goal is for the method to arrive at a very good approximation to `x` using a much smaller subspace $\mathcal{K}_m$. By default, the GMRES method selects $m = MIN(n, 10)$ but the user may specify a different value for m. The GMRES storage requirements grow as $O(n(m+1))$ and the number of flops grow as $O(4m^2 n - 4m^3/3)$.

In the below function `gsl_splinalg_itersolve_iterate()`, one GMRES iteration is defined as projecting the approximate solution vector onto each Krylov subspace $\mathcal{K}_1, ..., \mathcal{K}_m$, and so m can be kept smaller by "restarting" the method and calling `gsl_splinalg_itersolve_iterate()` multiple times, providing the updated approximation `x` to each new call. If the method is not adequately converging, the user may try increasing the parameter m.

GMRES is considered a robust general purpose iterative solver, however there are cases where the method stagnates if the matrix is not positive-definite and fails to reduce the residual until the very last projection onto the subspace $\mathcal{K}_n = \mathbf{R}^n$. In these cases, preconditioning the linear system can help, but GSL does not currently provide any preconditioners.

Iterating the Sparse Linear System

The following functions are provided to allocate storage for the sparse linear solvers and iterate the system to a solution.

gsl_splinalg_itersolve * **`gsl_splinalg_itersolve_alloc`**(const gsl_splinalg_itersolve_type * *T*, const size_t *n*, const size_t *m*)

This function allocates a workspace for the iterative solution of `n`-by-`n` sparse matrix systems. The iterative solver type is specified by `T`. The argument `m` specifies the size of the solution candidate subspace $\mathcal{K}_m$. The dimension `m` may be set to 0 in which case a reasonable default value is used.

void **gsl_splinalg_itersolve_free**(gsl_splinalg_itersolve * *w*)

This function frees the memory associated with the workspace `w`.

const char * **gsl_splinalg_itersolve_name**(const gsl_splinalg_itersolve * *w*)

This function returns a string pointer to the name of the solver.

int **gsl_splinalg_itersolve_iterate**(const gsl_spmatrix * *A*, const gsl_vector * *b*, const double *tol*, gsl_vector * *x*, gsl_splinalg_itersolve * *w*)

This function performs one iteration of the iterative method for the sparse linear system specfied by the matrix `A`, right hand side vector `b` and solution vector `x`. On input, `x` must be set to an initial guess for the solution. On output, `x` is updated to give the current solution estimate. The parameter `tol` specifies the relative tolerance between the residual norm and norm of `b` in order to check for convergence. When the following condition is satisfied:

$$||Ax - b|| \leq tol \times ||b||$$

the method has converged, the function returns `GSL_SUCCESS` and the final solution is provided in `x`. Otherwise, the function returns `GSL_CONTINUE` to signal that more iterations are required. Here, $|| \cdot ||$ represents the Euclidean norm. The input matrix `A` may be in triplet or compressed format.

double **gsl_splinalg_itersolve_normr**(const gsl_splinalg_itersolve * *w*)

This function returns the current residual norm $||r|| = ||Ax - b||$, which is updated after each call to `gsl_splinalg_itersolve_iterate()`.

Examples

This example program demonstrates the sparse linear algebra routines on the solution of a simple 1D Poisson equation on $[0, 1]$:

$$\frac{d^2u(x)}{dx^2} = f(x) = -\pi^2 \sin{(\pi x)}$$

with boundary conditions $u(0) = u(1) = 0$. The analytic solution of this simple problem is $u(x) = \sin \pi x$. We will solve this problem by finite differencing the left hand side to give

$$\frac{1}{h^2} \left(u_{i+1} - 2u_i + u_{i-1} \right) = f_i$$

Defining a grid of N points, $h = 1/(N-1)$. In the finite difference equation above, $u_0 = u_{N-1} = 0$ are known from the boundary conditions, so we will only put the equations for $i = 1, \ldots, N-2$ into the matrix system. The resulting $(N-2) \times (N-2)$ matrix equation is

$$\frac{1}{h^2}\begin{pmatrix} -2 & 1 & 0 & 0 & \ldots & 0 \\ 1 & -2 & 1 & 0 & \ldots & 0 \\ 0 & 1 & -2 & 1 & \ldots & 0 \\ \vdots & \vdots & \ddots & \ddots & \ddots & \vdots \\ 0 & \ldots & \ldots & 1 & -2 & 1 \\ 0 & \ldots & \ldots & \ldots & 1 & -2 \end{pmatrix} \begin{pmatrix} u_1 \\ u_2 \\ u_3 \\ \vdots \\ u_{N-3} \\ u_{N-2} \end{pmatrix} = \begin{pmatrix} f_1 \\ f_2 \\ f_3 \\ \vdots \\ f_{N-3} \\ f_{N-2} \end{pmatrix}$$

An example program which constructs and solves this system is given below. The system is solved using the iterative GMRES solver. Here is the output of the program:

```
iter 0 residual = 4.297275996844e-11
Converged
```

showing that the method converged in a single iteration. The calculated solution is shown in Fig. 40.

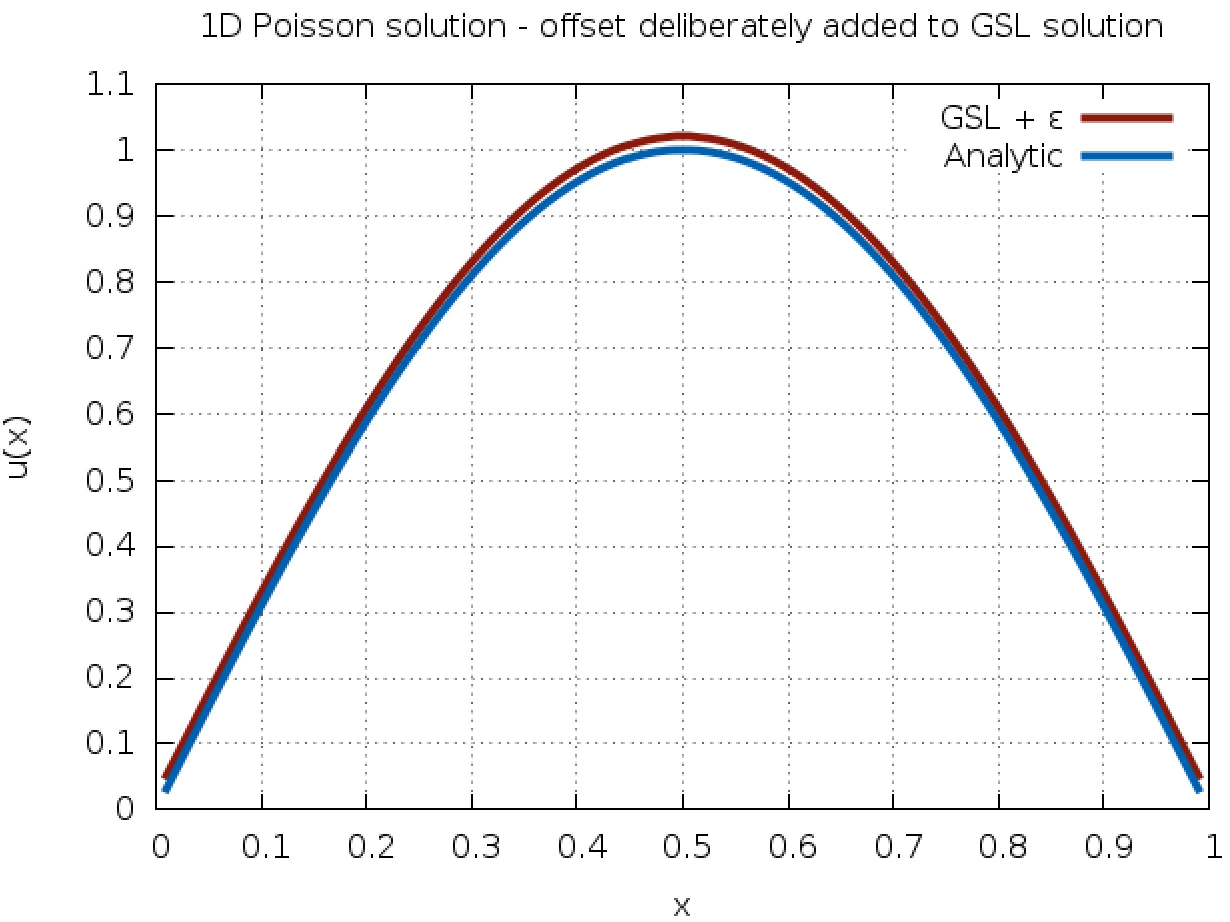

Fig. 40 Solution of PDE

```
#include <stdio.h>
#include <stdlib.h>
#include <math.h>

#include <gsl/gsl_math.h>
#include <gsl/gsl_vector.h>
#include <gsl/gsl_spmatrix.h>
#include <gsl/gsl_splinalg.h>

int
main()
{
  const size_t N = 100;                       /* number of grid points */
  const size_t n = N - 2;                     /* subtract 2 to exclude boundaries */
  const double h = 1.0 / (N - 1.0);           /* grid spacing */
  gsl_spmatrix *A = gsl_spmatrix_alloc(n ,n); /* triplet format */
  gsl_spmatrix *C;                            /* compressed format */
  gsl_vector *f = gsl_vector_alloc(n);        /* right hand side vector */
  gsl_vector *u = gsl_vector_alloc(n);        /* solution vector */
  size_t i;

  /* construct the sparse matrix for the finite difference equation */

  /* construct first row */
  gsl_spmatrix_set(A, 0, 0, -2.0);
  gsl_spmatrix_set(A, 0, 1, 1.0);

  /* construct rows [1:n-2] */
  for (i = 1; i < n - 1; ++i)
    {
      gsl_spmatrix_set(A, i, i + 1, 1.0);
      gsl_spmatrix_set(A, i, i, -2.0);
      gsl_spmatrix_set(A, i, i - 1, 1.0);
    }

  /* construct last row */
  gsl_spmatrix_set(A, n - 1, n - 1, -2.0);
  gsl_spmatrix_set(A, n - 1, n - 2, 1.0);

  /* scale by h^2 */
  gsl_spmatrix_scale(A, 1.0 / (h * h));

  /* construct right hand side vector */
  for (i = 0; i < n; ++i)
    {
      double xi = (i + 1) * h;
      double fi = -M_PI * M_PI * sin(M_PI * xi);
      gsl_vector_set(f, i, fi);
    }

  /* convert to compressed column format */
  C = gsl_spmatrix_ccs(A);

  /* now solve the system with the GMRES iterative solver */
  {
    const double tol = 1.0e-6;  /* solution relative tolerance */
    const size_t max_iter = 10; /* maximum iterations */
    const gsl_splinalg_itersolve_type *T = gsl_splinalg_itersolve_gmres;
    gsl_splinalg_itersolve *work =
      gsl_splinalg_itersolve_alloc(T, n, 0);
    size_t iter = 0;
    double residual;
    int status;

    /* initial guess u = 0 */
    gsl_vector_set_zero(u);

    /* solve the system A u = f */
    do
      {
        status = gsl_splinalg_itersolve_iterate(C, f, tol, u, work);
```

```
        /* print out residual norm ||A*u - f|| */
        residual = gsl_splinalg_itersolve_normr(work);
        fprintf(stderr, "iter %zu residual = %.12e\n", iter, residual);

        if (status == GSL_SUCCESS)
          fprintf(stderr, "Converged\n");
      }
    while (status == GSL_CONTINUE && ++iter < max_iter);

    /* output solution */
    for (i = 0; i < n; ++i)
      {
        double xi = (i + 1) * h;
        double u_exact = sin(M_PI * xi);
        double u_gsl = gsl_vector_get(u, i);

        printf("%f %.12e %.12e\n", xi, u_gsl, u_exact);
      }

    gsl_splinalg_itersolve_free(work);
  }

  gsl_spmatrix_free(A);
  gsl_spmatrix_free(C);
  gsl_vector_free(f);
  gsl_vector_free(u);

  return 0;
} /* main() */
```

References and Further Reading

The implementation of the GMRES iterative solver closely follows the publications

- H. F. Walker, Implementation of the GMRES method using Householder transformations, SIAM J. Sci. Stat. Comput. 9(1), 1988.
- Y. Saad, Iterative methods for sparse linear systems, 2nd edition, SIAM, 2003.