A Sparse Matrix Solver

`CGSolver.cpp` :

`CGSolver.cpp` is a C++ function that uses conjugate gradient algorithm to solve a linear system equations $Ax = b$ iteratively where A is in CSR format. The solver runs a maximum number of iterations equal to the size of the linear system. Function returns the number of iterations held to converge the solution to the specified tolerance, or -1 if the solver did not converge.

It accepts 5 inputs:

- `vector<double> &val` : A vector which contains the nonzero values of matrix A in left-to-right top-to-bottom order,
- `vector<int> &row_ptr` : A vector which contains the number of nonzero elements on the $(i-1)^{th}$ row in the original matrix,
- `vector<int> &col_idx` : A vector which contains the column index of each nonzero element of A,
- `vector<double> &b`, : Right hand side vector
- `vector<double> &x`, : Initial guess for solution
- `double tol` : Tolerance for how close solution needs to be

The initial guess is updated in-place as the solution and the other inputs are unchanged.

Tha algorithm is as follows:

Algorithm 1: Conjugate Gradient

Data: A as CSR Matrix, x as initial guess and b as RHS for $Ax = b$
Result: Number of iterations took for updating x s.t $|r| < tol$ for $r = Ax - b$
begin
  $Ax \longleftarrow A \times x$
  $r_0 \longleftarrow Ax - b$
  $p_0 \longleftarrow r_0$
  Initialize $x_{n+1}, r_{n+1}, p_{n+1}$
  `niter`=0
  `nitermax`= size of linear system
  while $niter \leq nitermax$ **do**
    `niter++`
    $\alpha = \frac{(r_n^T r_n)}{(p_n^T A p_n)}$
    $x_{n+1} = x_n + \alpha_n p_n$
    $r_{n+1} = r_n - \alpha_n A p_n$
    if $|r_{n+1}| < tol$ **then**
      $x \longleftarrow x_{n+1}$
      break while loop
    else
      $\beta_n = (r_{n+1}^T r_{n+1})/(r_n^T r_n)$
      $p_{n+1} = r_{n+1} + \beta_n p_n$
  return `niter`;

`matvecops.cpp` :

`CGSolver.cpp` uses functions provided in `matvecops.cpp` which are

- $A \times b$
- $A^T \times b$
- $a - b$
- $a + b$
- $|a|$: L^2 norm
- $a^T b$
- $\beta \times a$

Where A is a CSR matrix, a and b are vectors, and β is a scalar. These functions increase the modularity of CGSolver, avoiding repetitive lines.