Solvex converts a non-homogeneous linear system of equations Ax=b, A[m,n], x[n], b[m] into its equivalent linear program:

$$Min \sum_{i=1}^{m} \left(z_i^+ - z_i^- \right)$$

$$s.t.$$

$$a_{11}\left(x_1^+ - x_z^-\right) + a_{12}\left(x_2^+ - x_2^-\right) + ... + a_{1n}\left(x_n^+ - x_n^-\right) + \left(z_1^+ - z_z^-\right) = b_1$$

$$a_{21}\left(x_1^+ - x_z^-\right) + a_{22}\left(x_2^+ - x_2^-\right) + ... + a_{2n}\left(x_n^+ - x_n^-\right) + \left(z_2^+ - z_2^-\right) = b_2$$

$$...$$

$$a_{m1}\left(x_1^+ - x_z^-\right) + a_{m2}\left(x_2^+ - x_2^-\right) + ... + a_{mn}\left(x_n^+ - x_n^-\right) + \left(z_m^+ - z_m^-\right) = b_m$$

$$x_i \geq 0, z_i \geq 0$$

And solve the system using simple dense matrix simplex algorithm.

Usage:

1. Define an object of size m,n (named lp):
   ```
   solvex lp(m,n);
   ```

2. Add equations:
   ```
   lp.add_equ(double *A[0], double b[0]);
   ...
   lp.add_equ(double *A[m-1], double b[m-1]);
   ```

3. Solve System:
   ```
   lp.solve();
   ```

4. Get Solution:
   ```
   x[0] = lp.get_var(0);
   ...
   x[n] = lp.get_var(n-1);
   ```

Return Value:
add_equ returns 0 on success and -1 if trying to exceed matrix size (too many calles to add_equ).

solve() return RET_OK on success, RET_FAIL if the system could not be solved and RET_DIVZ if a division by zero is unavoidable, this is usually due to numerical instabilities.

Notes:

This code usually works well, however since I have not implemented matrix re-scaling and other numerical techniques needed for numerical stability - it may have scale problems in ill-conditioned systems. Such systems can still be solved using, for example IBM's linear programming solvers - however the interface is quite not as simple.

The objective function value can be obtained by calling **double lp.get_obj();**

lp.get_var(n) ... lp.get_var(2n-1); return the error value (Z^+-Z^-) for each equation.

The file solvex.cpp contains the following example main program (enabled by -DTEST flag at compile time) this examples solves a simple 3x3 system.

```
#include <stdio.h>
#include "solvex.h"

#ifdef TEST

//---------------------------------------------------------------
// The following code tests the solver using a simple 3x3 problem,
// It also demonstrates its usage
//---------------------------------------------------------------

main()
{
  double e1[] = {2, -5, 9};
  double e2[] = {7, 12, 3};
  double e3[] = {11, -4, 6};

  double X1, X2, X3;

  // Defind problem size 3 eqations and 3 variables
  solvex lp(3,3);

  // add equations
  lp.add_equ(e1,18); // e1(1)X1 + e1(2)X2 + e1(3)X3 = 18
  lp.add_equ(e2,-6);
  lp.add_equ(e3,2);

  // Solve the system, the return value should be zero is no error occurred
  printf("%d\n",lp.solve());

  // Print the objective function (the L1 minimum error)
  printf("Obj: %g\n",lp.get_obj());

  // These are the solutions
  printf("X0 = %g\n",X1=lp.get_var(0));
  printf("X1 = %g\n",X2=lp.get_var(1));
  printf("X2 = %g\n",X3=lp.get_var(2));

  // These are the error values for each equation
  printf("E1 = %g\n",lp.get_var(3));
  printf("E2 = %g\n",lp.get_var(4));
  printf("E3 = %g\n",lp.get_var(5));

  // Verify the solution
  printf("Verify1: %g\n", e1[0]*X1 + e1[1]*X2 +e1[2]*X3 - 18);
  printf("Verify2: %g\n", e2[0]*X1 + e2[1]*X2 +e2[2]*X3  + 6);
  printf("Verify3: %g\n", e3[0]*X1 + e3[1]*X2 +e3[2]*X3 - 2);

  return 0;
```