```cpp
// CPPLinearAlgebraIM.cpp : Defines the entry point for the application.
// (c) Tuesday, January 21, 2025, by James Pate Williams, Jr.
// Reference: "Elementary Numerical Analysis an Algorithmic Approach"
// Third Edition (c) 1980 by S. D. Conte and Carl de Boor
// Chapter 5 pages 230 - 231 Gauss-Seidel and Successive Overrelaxation
// Chapter 3 for Gsassian Elimination and LU-Decomposition
// https://www3.nd.edu/~zxu2/acms40390F12/Lec-7.3.pdf

#include "framework.h"
#include "CPPLinearAlgebraIM.h"
#include <chrono>
#include <cwchar>
#include <vector>

#define MAX_LOADSTRING 100

// resource definitions

#define IDC_EDIT1 201
#define IDC_EDIT2 202
#define IDC_EDIT3 203
#define IDC_EDIT4 204
#define IDC_EDIT5 205

#define IDC_STATIC1 301
#define IDC_STATIC2 302
#define IDC_STATIC3 303
#define IDC_STATIC4 304

#define IDC_BUTTON_COMPUTE 401
#define IDC_BUTTON_CLEAR 402

// Global Variables:

HINSTANCE hInst;                                // current instance
WCHAR szTitle[MAX_LOADSTRING];                  // The title bar text
WCHAR szWindowClass[MAX_LOADSTRING];            // the main window class name
WCHAR buffer[16384] = { }, line[128] = { };
std::vector<double> b;              // right-hand sides
std::vector<double> x0;             // solution iterate
std::vector<double> x1;             // solution vector
std::vector<std::vector<double>> A; // example matrix

// exercise Exercises 5.3-7 page 234 of reference

double omega1 = 1.6, omega2 = 1.2;
double a1[4][4] = {
    { 4.0, -1.0, 0.0, 0.0 }, { -1.0, 4.0, -1.0, 0.0 },
    { 0.0, -1.0, 4.0, -1.0 }, { 0.0, 0.0, -1.0, 4.0 } };
double a2[3][3] = {
    { 5.0, -2.0, 3.0}, { -3.0, 9.0, 1.0 },
    { 2.0, -1.0, -7.0 } };
```

```
 53 double b1[4] = { 1.0, 1.0, 1.0, 1.0 };
 54 double b2[3] = { -1.0, 2.0, 3.0 };
 55
 56 // Forward declarations of functions included in this code module:
 57
 58 ATOM                MyRegisterClass(HINSTANCE hInstance);
 59 BOOL                InitInstance(HINSTANCE, int);
 60 LRESULT CALLBACK    WndProc(HWND, UINT, WPARAM, LPARAM);
 61 INT_PTR CALLBACK    About(HWND, UINT, WPARAM, LPARAM);
 62
 63 int APIENTRY wWinMain(_In_ HINSTANCE hInstance,
 64                       _In_opt_ HINSTANCE hPrevInstance,
 65                       _In_ LPWSTR    lpCmdLine,
 66                       _In_ int       nCmdShow)
 67 {
 68     UNREFERENCED_PARAMETER(hPrevInstance);
 69     UNREFERENCED_PARAMETER(lpCmdLine);
 70
 71     // TODO: Place code here.
 72
 73     // Initialize global strings
 74     LoadStringW(hInstance, IDS_APP_TITLE, szTitle, MAX_LOADSTRING);
 75     LoadStringW(hInstance, IDC_CPPLINEARALGEBRAIM, szWindowClass,
          MAX_LOADSTRING);
 76     MyRegisterClass(hInstance);
 77
 78     // Perform application initialization:
 79     if (!InitInstance (hInstance, nCmdShow))
 80     {
 81         return FALSE;
 82     }
 83
 84     HACCEL hAccelTable = LoadAccelerators(hInstance, MAKEINTRESOURCE
          (IDC_CPPLINEARALGEBRAIM));
 85
 86     MSG msg;
 87
 88     // Main message loop:
 89     while (GetMessage(&msg, nullptr, 0, 0))
 90     {
 91         if (!TranslateAccelerator(msg.hwnd, hAccelTable, &msg))
 92         {
 93             TranslateMessage(&msg);
 94             DispatchMessage(&msg);
 95         }
 96     }
 97
 98     return (int) msg.wParam;
 99 }
100 //
101 //  FUNCTION: MyRegisterClass()
102 //
```

```
103 //  PURPOSE: Registers the window class.
104 //
105 ATOM MyRegisterClass(HINSTANCE hInstance)
106 {
107     WNDCLASSEXW wcex = { };
108
109     wcex.cbSize = sizeof(WNDCLASSEX);
110
111     wcex.style          = CS_HREDRAW | CS_VREDRAW;
112     wcex.lpfnWndProc    = WndProc;
113     wcex.cbClsExtra     = 0;
114     wcex.cbWndExtra     = 0;
115     wcex.hInstance      = hInstance;
116     wcex.hIcon          = LoadIcon(hInstance, MAKEINTRESOURCE
          (IDI_CPPLINEARALGEBRAIM));
117     wcex.hCursor        = LoadCursor(nullptr, IDC_ARROW);
118     wcex.hbrBackground  = (HBRUSH)(COLOR_WINDOW+1);
119     wcex.lpszMenuName   = MAKEINTRESOURCEW(IDC_CPPLINEARALGEBRAIM);
120     wcex.lpszClassName  = szWindowClass;
121     wcex.hIconSm        = LoadIcon(wcex.hInstance, MAKEINTRESOURCE(IDI_SMALL));
122
123     return RegisterClassExW(&wcex);
124 }
125 //
126 //   FUNCTION: InitInstance(HINSTANCE, int)
127 //
128 //   PURPOSE: Saves instance handle and creates main window
129 //
130 //   COMMENTS:
131 //
132 //        In this function, we save the instance handle in a global variable and
133 //        create and display the main program window.
134 //
135 BOOL InitInstance(HINSTANCE hInstance, int nCmdShow)
136 {
137    hInst = hInstance; // Store instance handle in our global variable
138
139    HWND hWnd = CreateWindowW(szWindowClass, szTitle, WS_OVERLAPPEDWINDOW,
140       CW_USEDEFAULT, 0, CW_USEDEFAULT, 0, nullptr, nullptr, hInstance, nullptr);
141
142    if (!hWnd)
143    {
144       return FALSE;
145    }
146
147    ShowWindow(hWnd, nCmdShow);
148    UpdateWindow(hWnd);
149
150    return TRUE;
151 }
152
153 void GaussSeidel(
```

```
154     double tolerance, double dummy,
155     int maxIts, int n,
156     double& norm2x, int& its,
157     std::vector<double>& b,
158     std::vector<double>& x0,
159     std::vector<double>& x1,
160     std::vector<std::vector<double>>& A)
161 {
162     int i = 0, j = 0;
163     std::vector<double> c;              // right-hand sides
164     std::vector<std::vector<double>> B; // working storage
165 
166     c.resize(n);
167     B.resize(n);
168 
169     for (auto& row : B)
170     {
171         row.resize(n);
172     }
173 
174     for (i = 0; i < n; i++)
175     {
176         x1[i] = x0[i];
177     }
178 
179     for (i = 0; i < n; i++)
180     {
181         for (j = 0; j < n; j++)
182         {
183             if (i != j)
184             {
185                 B[i][j] = -A[i][j] / A[i][i];
186             }
187 
188             else
189             {
190                 B[i][j] = 0.0;
191             }
192         }
193 
194         c[i] = b[i] / A[i][i];
195     }
196 
197     its = 0;
198 
199     while (its < maxIts)
200     {
201         for (i = 0; i < n; i++)
202         {
203             double sum1 = 0;
204 
205             if ((i - 1) >= 0)
```

```
206             {
207                 for (j = 0; j <= i - 1; j++)
208                 {
209                     sum1 += B[i][j] * x1[j];
210                 }
211             }
212 
213             double sum2 = 0;
214 
215             if ((i + 1) < n)
216             {
217                 for (j = i + 1; j < n; j++)
218                 {
219                     sum2 += B[i][j] * x0[j];
220                 }
221             }
222 
223             x1[i] = sum1 + sum2 + c[i];
224         }
225 
226         norm2x = 0.0;
227 
228         for (i = 0; i < n; i++)
229         {
230             double delta = x1[i] - x0[i];
231 
232             norm2x += delta * delta;
233         }
234 
235         norm2x = sqrt(norm2x);
236 
237         its++;
238 
239         if (norm2x < tolerance)
240         {
241             break;
242         }
243 
244         for (i = 0; i < n; i++)
245         {
246             x0[i] = x1[i];
247         }
248     }
249 }
250 
251 void SuccessOverrelaxation(
252     double tolerance, double omega,
253     int maxIts, int n,
254     double& norm2x, int& its,
255     std::vector<double>& b,
256     std::vector<double>& x0,
257     std::vector<double>& x1,
```

```
258     std::vector<std::vector<double>>& A)
259 {
260     int i = 0, j = 0;
261 
262     for (i = 0; i < n; i++)
263     {
264         x1[i] = x0[i];
265     }
266 
267     its = 0;
268 
269     while (its < maxIts)
270     {
271         for (i = 0; i < n; i++)
272         {
273             double sum1 = 0;
274 
275             if ((i - 1) >= 0)
276             {
277                 for (j = 0; j < i; j++)
278                 {
279                     sum1 += A[i][j] * x1[j];
280                 }
281             }
282 
283             double sum2 = 0;
284 
285             if ((i + 1) < n)
286             {
287                 for (j = i + 1; j < n; j++)
288                 {
289                     sum2 += A[i][j] * x0[j];
290                 }
291             }
292 
293             x1[i] = (1.0 - omega) * x0[i] +
294                 omega * (b[i] - sum1 - sum2) / A[i][i];
295         }
296 
297         norm2x = 0.0;
298 
299         for (i = 0; i < n; i++)
300         {
301             double delta = x1[i] - x0[i];
302 
303             norm2x += delta * delta;
304         }
305 
306         norm2x = sqrt(norm2x);
307 
308         its++;
309 
```

```
310         if (norm2x < tolerance)
311         {
312             break;
313         }
314
315         for (i = 0; i < n; i++)
316         {
317             x0[i] = x1[i];
318         }
319     }
320 }
321
322 void Jacobi(
323     double tolerance, double dummy,
324     int maxIts, int n,
325     double& norm2x, int& its,
326     std::vector<double>& b,
327     std::vector<double>& x0,
328     std::vector<double>& x1,
329     std::vector<std::vector<double>>& A)
330 {
331     its = 0;
332
333     while (its < maxIts)
334     {
335         for (int i = 0; i < n; i++)
336         {
337             double sum = 0.0;
338
339             for (int j = 0; j < n; j++)
340             {
341                 if (j != i)
342                 {
343                     sum += A[i][j] * x0[j];
344                 }
345             }
346
347             x1[i] = (b[i] - sum) / A[i][i];
348         }
349
350         norm2x = 0.0;
351
352         for (int i = 0; i < n; i++)
353         {
354             double delta = x1[i] - x0[i];
355
356             norm2x += delta * delta;
357         }
358
359         norm2x = sqrt(norm2x);
360
361         its++;
```

```
362
363         if (norm2x < tolerance)
364         {
365             break;
366         }
367
368         for (int i = 0; i < n; i++)
369         {
370             x0[i] = x1[i];
371         }
372     }
373 }
374
375 void Substitute(
376     int n,
377     std::vector<double>& b,
378     std::vector<double>& x1,
379     std::vector<std::vector<double>>& w,
380     std::vector<int>& pivot)
381 {
382     double sum;
383     int i, j, n1 = n - 1;
384
385     if (n == 1)
386     {
387         x1[0] = b[0] / w[0][0];
388         return;
389     }
390
391     // forward substitution
392
393     x1[0] = b[pivot[0]];
394
395     for (i = 1; i < n; i++)
396     {
397         for (j = 0, sum = 0; j < i; j++)
398             sum += w[i][j] * x1[j];
399
400         x1[i] = b[pivot[i]] - sum;
401     }
402
403     // backward substitution
404
405     x1[n1] /= w[n1][n1];
406
407     for (i = n - 2; i >= 0; i--)
408     {
409         for (j = i + 1, sum = 0; j < n; j++)
410             sum += w[i][j] * x1[j];
411
412         x1[i] = (x1[i] - sum) / w[i][i];
413     }
```

```
414 }
415
416 void GaussianElimination(
417     double tolerance, double dummy,
418     int maxIts, int n,
419     double& norm2x, int& its,
420     std::vector<double>& b,
421     std::vector<double>& x0,
422     std::vector<double>& x1,
423     std::vector<std::vector<double>>& A)
424 {
425
426     // returns false if matrix is singular
427
428     double awikod, col_max, ratio, row_max, temp;
429     int flag = 1, i, i_star, j, k;
430     std::vector<double> d(n);
431     std::vector<std::vector<double>> w;
432     std::vector<int> pivot(n);
433
434     w.resize(n);
435
436     for (auto& row : w)
437     {
438         row.resize(n);
439     }
440
441     for (i = 0; i < n; i++)
442     {
443         for (j = 0; j < n; j++)
444         {
445             w[i][j] = A[i][j];
446         }
447     }
448
449     for (i = 0; i < n; i++)
450     {
451         pivot[i] = i;
452         row_max = 0;
453
454         for (j = 0; j < n; j++)
455             row_max = fmax(row_max, fabs(w[i][j]));
456
457         if (row_max == 0)
458         {
459             flag = 0;
460             row_max = 1;
461         }
462
463         d[i] = row_max;
464     }
465
```

```
466     if (n <= 1)
467     {
468         its = -1;
469         return;
470     }
471
472     // factorization
473
474     for (k = 0; k < n - 1; k++)
475     {
476
477         // determine pivot row the row i_star
478
479         col_max = fabs(w[k][k]) / d[k];
480         i_star = k;
481         for (i = k + 1; i < n; i++)
482         {
483             awikod = fabs(w[i][k]) / d[i];
484
485             if (awikod > col_max)
486             {
487                 col_max = awikod;
488                 i_star = i;
489             }
490         }
491
492         if (col_max == 0)
493             flag = 0;
494
495         else
496         {
497             if (i_star > k)
498             {
499
500                 // make k the pivot row by
501                 // interchanging with i_star
502
503                 flag *= -1;
504                 i = pivot[i_star];
505                 pivot[i_star] = pivot[k];
506                 pivot[k] = i;
507                 temp = d[i_star];
508                 d[i_star] = d[k];
509                 d[k] = temp;
510
511                 for (j = 0; j < n; j++)
512                 {
513                     temp = w[i_star][j];
514                     w[i_star][j] = w[k][j];
515                     w[k][j] = temp;
516                 }
517             }
```

```
518
519             // eliminate x[k]
520
521             for (i = k + 1; i < n; i++)
522             {
523                 w[i][k] /= w[k][k];
524                 ratio = w[i][k];
525
526                 for (j = k + 1; j < n; j++)
527                     w[i][j] -= ratio * w[k][j];
528             }
529         }
530     }
531
532     if (w[n - 1][n - 1] == 0)
533     {
534         flag = 0;
535     }
536
537     if (flag == 0)
538     {
539         its = -1;
540     }
541
542     else
543     {
544         Substitute(n, b, x1, w, pivot);
545         its = 1;
546     }
547 }
548
549 void LUDecomposition(
550     double tolerance, double dummy,
551     int maxIts, int n,
552     double& norm2x, int& its,
553     std::vector<double>& b,
554     std::vector<double>& x0,
555     std::vector<double>& x1,
556     std::vector<std::vector<double>>& A)
557 {
558     std::vector<double> y(n, 0.0);
559     std::vector<std::vector<double>> l(
560         n, std::vector<double>(n));
561     std::vector<std::vector<double>> u(
562         n, std::vector<double>(n));
563     int n1 = n - 1;
564
565     its = 0;
566
567     for (int i = 0; i < n; i++)
568     {
569         x1[i] = 0.0;
```

```
570
571         for (int j = 0; j < n; j++)
572         {
573             l[i][j] = u[i][j] = 0.0;
574         }
575     }
576
577     for (int k = 0; k < n; k++)
578     {
579         for (int j = k; j < n; j++)
580         {
581             double sum = 0.0;
582
583             if (k >= 1)
584             {
585                 for (int p = 0; p <= k - 1; p++)
586                 {
587                     sum += l[k][p] * u[p][j];
588                 }
589             }
590
591             u[k][j] = A[k][j] - sum;
592         }
593
594         for (int i = k + 1; i < n; i++)
595         {
596             double sum = 0.0;
597
598             if (k >= 1)
599             {
600                 for (int p = 0; p <= k - 1; p++)
601                 {
602                     sum += l[i][p] * u[p][k];
603                 }
604             }
605
606             l[i][k] = (A[i][k] - sum) / u[k][k];
607         }
608     }
609
610     // forward substitution
611
612     for (int k = 0; k < n; k++)
613     {
614         double sum = 0.0;
615
616         for (int j = 0; j < k; j++)
617         {
618             sum += l[k][j] * y[j];
619         }
620
621         y[k] = b[k] - sum;
```

```
622     }
623
624     // backward substitution
625
626     for (int k = n - 1; k >= 0; k--)
627     {
628         double sum = 0;
629
630         if (k + 1 < n)
631         {
632             for (int j = k + 1; j < n; j++)
633             {
634                 sum += u[k][j] * x1[j];
635             }
636         }
637
638         x1[k] = (y[k] - sum) / u[k][k];
639     }
640 }
641
642 void Test(
643     double tolerance, double dummy,
644     int maxIts, int n, int problem,
645     double& norm2b,
646     double& norm2x, int& its,
647     std::vector<double>& b,
648     std::vector<double>& x0,
649     std::vector<double>& x1,
650     std::vector<std::vector<double>>& A,
651     void (*function)(double, double,
652         int, int, double&, int&,
653         std::vector<double>&,
654         std::vector<double>&,
655         std::vector<double>&,
656         std::vector<std::vector<double>>&))
657 {
658     int i, j;
659
660     for (i = 0; i < n; i++)
661     {
662         if (n == 4 && problem == 1 || problem == 3 ||
663             problem == 5 || problem == 7 || problem == 9)
664         {
665             b[i] = b1[i];
666         }
667
668         else if (n == 3 && problem == 2 || problem == 4 ||
669             problem == 6 || problem == 8 || problem == 10)
670         {
671             b[i] = b2[i];
672         }
673
```

```
674         x0[i] = 0.0;
675
676         for (j = 0; j < n; j++)
677         {
678             if (n == 4 && problem == 1 || problem == 3 ||
679                 problem == 5 || problem == 7 || problem == 9)
680             {
681                 A[i][j] = a1[i][j];
682             }
683
684             else if (n == 3 && problem == 2 || problem == 4 ||
685                 problem == 6 || problem == 8 || problem == 10)
686             {
687                 A[i][j] = a2[i][j];
688             }
689         }
690     }
691
692     function(tolerance, dummy, maxIts, n,
693         norm2x, its, b, x0, x1, A);
694
695     norm2b = 0.0;
696
697     for (i = 0; i < n; i++)
698     {
699         double sum = 0.0;
700
701         for (j = 0; j < n; j++)
702         {
703             sum += A[i][j] * x1[j];
704         }
705
706         norm2b += pow(b[i] - sum, 2.0);
707     }
708
709     norm2b = sqrt(norm2b);
710 }
711
712 //
713 //  FUNCTION: WndProc(HWND, UINT, WPARAM, LPARAM)
714 //
715 //  PURPOSE: Processes messages for the main window.
716 //
717 //  WM_COMMAND  - process the application menu
718 //  WM_PAINT    - Paint the main window
719 //  WM_DESTROY  - post a quit message and return
720 //
721 //
722 LRESULT CALLBACK WndProc(HWND hWnd, UINT message, WPARAM wParam, LPARAM lParam)
723 {
724     static double norm2b = 0, norm2x = 0, tolerance = 0;
725     static int its = 0, maxIts = 0, problem;
```

```
726     static size_t i = 0, j = 0, n;
727     static WCHAR* endPt;
728     static HWND hwndEdit1 = 0, hwndEdit2 = 0;
729     static HWND hwndEdit3 = 0, hwndEdit4 = 0, hwndEdit5 = 0;
730
731     switch (message)
732     {
733     case WM_CREATE:
734     {
735         CreateWindowEx(0, L"STATIC", L"n (3 or 4):", WS_CHILD | WS_VISIBLE,
736             10, 10, 80, 20, hWnd, (HMENU)IDC_STATIC1, GetModuleHandle(NULL),
                NULL);
737         CreateWindowEx(0, L"STATIC", L"Max Its:", WS_CHILD | WS_VISIBLE,
738             10, 40, 80, 20, hWnd, (HMENU)IDC_STATIC2, GetModuleHandle(NULL),
                NULL);
739         CreateWindowEx(0, L"STATIC", L"Tolerance:", WS_CHILD | WS_VISIBLE,
740             10, 70, 80, 20, hWnd, (HMENU)IDC_STATIC3, GetModuleHandle(NULL),
                NULL);
741         CreateWindowEx(0, L"STATIC", L"Prob (1 - 4):", WS_CHILD | WS_VISIBLE,
742             10, 100, 80, 20, hWnd, (HMENU)IDC_STATIC4, GetModuleHandle(NULL),
                NULL);
743
744         hwndEdit1 = CreateWindowEx(0, L"EDIT", NULL, WS_CHILD | WS_VISIBLE |
              WS_BORDER,
745             100, 10, 200, 20, hWnd, (HMENU)IDC_EDIT1, GetModuleHandle(NULL),
                NULL);
746         hwndEdit2 = CreateWindowEx(0, L"EDIT", NULL, WS_CHILD | WS_VISIBLE |
              WS_BORDER,
747             100, 40, 200, 20, hWnd, (HMENU)IDC_EDIT2, GetModuleHandle(NULL),
                NULL);
748         hwndEdit3 = CreateWindowEx(0, L"EDIT", NULL, WS_CHILD | WS_VISIBLE |
              WS_BORDER,
749             100, 70, 200, 20, hWnd, (HMENU)IDC_EDIT3, GetModuleHandle(NULL),
                NULL);
750         hwndEdit4 = CreateWindowEx(0, L"EDIT", NULL, WS_CHILD | WS_VISIBLE |
              WS_BORDER,
751             100, 100, 200, 20, hWnd, (HMENU)IDC_EDIT4, GetModuleHandle(NULL),
                NULL);
752         hwndEdit5 = CreateWindowEx(0, L"EDIT", NULL, WS_CHILD | WS_VISIBLE |
              WS_BORDER
753             | ES_MULTILINE, 100, 170, 400, 400, hWnd, (HMENU)IDC_EDIT5,
754             GetModuleHandle(NULL), NULL);
755
756         CreateWindowEx(0, L"BUTTON", L"Compute", WS_CHILD | WS_VISIBLE |
              BS_PUSHBUTTON,
757             10, 130, 80, 30, hWnd, (HMENU)IDC_BUTTON_COMPUTE, GetModuleHandle
                (NULL), NULL);
758         CreateWindowEx(0, L"BUTTON", L"Clear", WS_CHILD | WS_VISIBLE |
              BS_PUSHBUTTON,
759             220, 130, 80, 30, hWnd, (HMENU)IDC_BUTTON_CLEAR, GetModuleHandle
                (NULL), NULL);
760
```

```
761         SetDlgItemText(hWnd, IDC_EDIT1, L"4");
762         SetDlgItemText(hWnd, IDC_EDIT2, L"100");
763         SetDlgItemText(hWnd, IDC_EDIT3, L"1.0e-12");
764         SetDlgItemText(hWnd, IDC_EDIT4, L"1");
765     }
766     break;
767     case WM_COMMAND:
768         {
769             int wmId = LOWORD(wParam);
770             // Parse the menu selections:
771             switch (wmId)
772             {
773             case IDC_BUTTON_COMPUTE:
774             {
775                 n = GetDlgItemInt(hWnd, IDC_EDIT1, FALSE, FALSE);
776 
777                 if (n != 3 && n != 4)
778                 {
779                     MessageBox(hWnd, L"n must be 3 or 4", L"Warning",
780                         MB_OK | MB_ICONWARNING);
781                     return 0;
782                 }
783 
784                 maxIts = GetDlgItemInt(hWnd, IDC_EDIT2, FALSE, FALSE);
785                 problem = GetDlgItemInt(hWnd, IDC_EDIT4, FALSE, FALSE);
786 
787                 if (problem < 1 || problem > 10)
788                 {
789                     MessageBox(hWnd, L"Problem must be 1 to 10", L"Warning",
790                         MB_OK | MB_ICONWARNING);
791                     return 0;
792                 }
793 
794                 GetWindowText(hwndEdit3, line, 127);
795                 tolerance = wcstod(line, &endPt);
796 
797                 auto start = std::chrono::high_resolution_clock::now();
798 
799                 x0.resize(n);
800                 x1.resize(n);
801                 b.resize(n);
802 
803                 A.resize(n);
804 
805                 for (auto& row : A)
806                 {
807                     row.resize(n);
808                 }
809 
810                 if (n == 4 && problem == 1)
811                 {
812                     Test(tolerance, omega1, maxIts, n, problem,
```

```
813                         norm2b, norm2x, its, b, x0, x1, A, GaussSeidel);
814                 }
815
816                 else if (n == 3 && problem == 2)
817                 {
818                     Test(tolerance, omega2, maxIts, n, problem,
819                         norm2b, norm2x, its, b, x0, x1, A, GaussSeidel);
820                 }
821
822                 else if (n == 4 && problem == 3)
823                 {
824                     Test(tolerance, omega1, maxIts, n, problem,
825                         norm2b, norm2x, its, b, x0, x1, A,
826                         SuccessOverrelaxation);
827                 }
828
829                 else if (n == 3 && problem == 4)
830                 {
831                     Test(tolerance, omega2, maxIts, n, problem,
832                         norm2b, norm2x, its, b, x0, x1, A,
833                         SuccessOverrelaxation);
834                 }
835
836                 else if (n == 4 && problem == 5)
837                 {
838                     Test(tolerance, omega1, maxIts, n, problem,
839                         norm2b, norm2x, its, b, x0, x1, A, Jacobi);
840                 }
841
842                 else if (n == 3 && problem == 6)
843                 {
844                     Test(tolerance, omega2, maxIts, n, problem,
845                         norm2b, norm2x, its, b, x0, x1, A, Jacobi);
846                 }
847
848                 else if (n == 4 && problem == 7)
849                 {
850                     Test(tolerance, omega1, maxIts, n, problem,
851                         norm2b, norm2x, its, b, x0, x1, A, GaussianElimination);
852                 }
853
854                 else if (n == 3 && problem == 8)
855                 {
856                     Test(tolerance, omega2, maxIts, n, problem,
857                         norm2b, norm2x, its, b, x0, x1, A, GaussianElimination);
858                 }
859
860                 else if (n == 4 && problem == 9)
861                 {
862                     Test(tolerance, omega1, maxIts, n, problem,
863                         norm2b, norm2x, its, b, x0, x1, A, LUDecomposition);
864                 }
```

```
865
866                 else if (n == 3 && problem == 10)
867                 {
868                     Test(tolerance, omega1, maxIts, n, problem,
869                         norm2b, norm2x, its, b, x0, x1, A, LUDecomposition);
870                 }
871
872                 else if (n == 4)
873                 {
874                     MessageBox(hWnd, L"If n = 4 Problem must be 1 or 3 or 5 or 7
                        or 9",
875                         L"Warning", MB_OK | MB_ICONWARNING);
876                     return 0;
877                 }
878
879                 else if (n == 3)
880                 {
881                     MessageBox(hWnd, L"If n = 3 Problem must be 2 or 4 or 6 or 8
                        or 10",
882                         L"Warning", MB_OK | MB_ICONWARNING);
883                     return 0;
884                 }
885
886                 auto endTm = std::chrono::high_resolution_clock::now();
887                 auto duration = std::chrono::duration_cast<
888                     std::chrono::microseconds>(endTm - start);
889                 long long runtime = duration.count();
890
891                 if (problem == 1 || problem == 2)
892                 {
893                     wsprintf(line, L"Gauss-Seidel\r\n\r\n");
894                     wcscat_s(buffer, 16383, line);
895                 }
896
897                 else if (problem == 3 || problem == 4)
898                 {
899                     wsprintf(line, L"Successive Overrelaxation (SOR) Method\r\n
                        \r\n");
900                     wcscat_s(buffer, 16383, line);
901                 }
902
903                 else if (problem == 5 || problem == 6)
904                 {
905                     wsprintf(line, L"Jacobi Method\r\n\r\n");
906                     wcscat_s(buffer, 16383, line);
907                 }
908
909                 else if (problem == 7 || problem == 8)
910                 {
911                     wsprintf(line, L"Gaussian Elimination\r\n\r\n");
912                     wcscat_s(buffer, 16383, line);
913                 }
```

```
914
915                 else if (problem == 9 || problem == 10)
916                 {
917                     wsprintf(line, L"LU Decomposition\r\n\r\n");
918                     wcscat_s(buffer, 16383, line);
919                 }
920
921                 wsprintf(line, L"Solution Vector x\r\n\r\n");
922                 wcscat_s(buffer, 16383, line);
923
924                 for (size_t i = 0; i < n; i++)
925                 {
926                     swprintf_s(line, L"%13.10lf\t", x1[i]);
927                     wcscat_s(buffer, 16383, line);
928                 }
929
930                 wsprintf(line, L"\r\n");
931                 wcscat_s(buffer, 16383, line);
932
933                 wsprintf(line, L"\r\nX - Norm - 2 = ");
934                 wcscat_s(buffer, 16383, line);
935
936                 swprintf_s(line, L"%13.10lf\r\n", norm2x);
937                 wcscat_s(buffer, 16383, line);
938
939                 wsprintf(line, L"\r\nb - Norm - 2 = ");
940                 wcscat_s(buffer, 16383, line);
941
942                 swprintf_s(line, L"%13.10lf\r\n\r\n", norm2b);
943                 wcscat_s(buffer, 16383, line);
944
945                 swprintf_s(line, L"Iterations = %ld\r\n\r\n", its);
946                 wcscat_s(buffer, 16383, line);
947
948                 swprintf_s(line, L"Runtime = %lld Microseconds\r\n\r\n",
949                     runtime);
950                 wcscat_s(buffer, 16383, line);
951
952                 SetDlgItemText(hWnd, IDC_EDIT5, buffer);
953             }
954             break;
955             case IDC_BUTTON_CLEAR:
956             {
957                 wcscpy_s(buffer, 16383, L"\0");
958                 SetDlgItemText(hWnd, IDC_EDIT5, buffer);
959             }
960             break;
961             case IDM_ABOUT:
962                 DialogBox(hInst, MAKEINTRESOURCE(IDD_ABOUTBOX), hWnd, About);
963                 break;
964             case IDM_EXIT:
965                 DestroyWindow(hWnd);
```

```
966                 break;
967             default:
968                 return DefWindowProc(hWnd, message, wParam, lParam);
969             }
970         }
971         break;
972     case WM_PAINT:
973         {
974             PAINTSTRUCT ps;
975             HDC hdc = BeginPaint(hWnd, &ps);
976             // TODO: Add any drawing code that uses hdc here...
977             EndPaint(hWnd, &ps);
978         }
979         break;
980     case WM_SIZE:
981         if (hwndEdit5)
982         {
983             RECT rcClient;
984             GetClientRect(hWnd, &rcClient);
985             SetWindowPos(hwndEdit5, NULL, 10, 170, rcClient.right - 20,
986                 rcClient.bottom - 20, SWP_NOZORDER);
987         }
988         break;
989     case WM_DESTROY:
990         PostQuitMessage(0);
991         break;
992     default:
993         return DefWindowProc(hWnd, message, wParam, lParam);
994     }
995     return 0;
996 }
997
998 // Message handler for about box.
999 INT_PTR CALLBACK About(HWND hDlg, UINT message, WPARAM wParam, LPARAM lParam)
1000 {
1001     UNREFERENCED_PARAMETER(lParam);
1002     switch (message)
1003     {
1004     case WM_INITDIALOG:
1005         return (INT_PTR)TRUE;
1006
1007     case WM_COMMAND:
1008         if (LOWORD(wParam) == IDOK || LOWORD(wParam) == IDCANCEL)
1009         {
1010             EndDialog(hDlg, LOWORD(wParam));
1011             return (INT_PTR)TRUE;
1012         }
1013         break;
1014     }
1015     return (INT_PTR)FALSE;
1016 }
```