```cpp
// AppoximationByOrthgonalPolynomials.cpp (c) Saturday,
// October 26, 2024 by James Pate Williams, Jr.
// Least-Squares Approximation by Orthogonal
// Polynomials Translated from a FORTRAN
// Subroutine found in "Numerical Analysis An
// Algorithmic Approach Third Edition (c) 1980 by
// S. D. Conte and Carl de Boor Chapter 6.4

#include "framework.h"
#include <limits.h>
#include <math.h>
#include <stdio.h>
#include <fstream>
#include <vector>
#include "AppoximationByOrthgonalPolynomials.h"
using namespace std;

#define MAX_LOADSTRING 100
#define MAX_POINTS 4096
#define MAX_DEGREE 16

typedef struct tagPoint2d
{
    double x, y;
} POINT2D, *PPOINT2D;

// Global Variables:
HINSTANCE hInst;                                // current instance
WCHAR szTitle[MAX_LOADSTRING];                  // The title bar text
WCHAR szWindowClass[MAX_LOADSTRING];            // the main window class name
double x0, x1;                                  // end-points of abscissas
double numberPoints;                            // number points
int degree;                                     // degree orthogonal polynomial
int npoints;                                    // number of fitting points
int function;                                   // 1 = graph f1, 2 = graph f2
vector<double> b(MAX_POINTS, 0);                // term of recurrence relation
vector<double> c(MAX_POINTS, 0);                // term of recurrence relation
vector<double> d(MAX_POINTS, 0);                // term of recurrence relation
vector<double> s(MAX_POINTS, 0);                // term of recurrence relation
vector<double> error(MAX_POINTS, 0);            // error vector
vector<double> f(MAX_POINTS, 0);                // function values
vector<double> pj(MAX_POINTS, 0);               // polynomial values
vector<double> pjm1(MAX_POINTS, 0);             // polynomial values minus 1
vector<double> w(MAX_POINTS, 0);                // ort poly weight values
vector<double> x(MAX_POINTS, 0);                // x values
vector<POINT2D> points0(MAX_POINTS);            // x, y values (2d) function
vector<POINT2D> points1(MAX_POINTS);            // x, y values (2d) approximation

// Forward declarations of functions included in this code module:
ATOM                MyRegisterClass(HINSTANCE hInstance);
BOOL                InitInstance(HINSTANCE, int);
LRESULT CALLBACK    WndProc(HWND, UINT, WPARAM, LPARAM);
```

```
53  INT_PTR CALLBACK    About(HWND, UINT, WPARAM, LPARAM);
54  INT_PTR CALLBACK    GetCurveFittingData(HWND, UINT, WPARAM, LPARAM);
55  INT_PTR CALLBACK    DrawGraph(HWND, UINT, WPARAM, LPARAM);
56
57  double f1(double x)
58  {
59      return x * sin(x);
60  }
61
62  double f2(double x)
63  {
64      return x * log(1.0 - x);
65  }
66
67  double f3(double x)
68  {
69      return exp(-x) / sqrt(4.0 * atan(1.0));
70  }
71
72  double wt(double x)
73  {
74      return 1.0;
75  }
76
77  int APIENTRY wWinMain(_In_ HINSTANCE hInstance,
78                        _In_opt_ HINSTANCE hPrevInstance,
79                        _In_ LPWSTR    lpCmdLine,
80                        _In_ int       nCmdShow)
81  {
82      UNREFERENCED_PARAMETER(hPrevInstance);
83      UNREFERENCED_PARAMETER(lpCmdLine);
84
85      // TODO: Place code here.
86
87      // Initialize global strings
88      LoadStringW(hInstance, IDS_APP_TITLE, szTitle, MAX_LOADSTRING);
89      LoadStringW(hInstance, IDC_APPOXIMATIONBYORTHGONALPOLYNOMIALS, szWindowClass, ↵
            MAX_LOADSTRING);
90      MyRegisterClass(hInstance);
91
92      // Perform application initialization:
93      if (!InitInstance (hInstance, nCmdShow))
94      {
95          return FALSE;
96      }
97
98      HACCEL hAccelTable = LoadAccelerators(hInstance, MAKEINTRESOURCE                ↵
            (IDC_APPOXIMATIONBYORTHGONALPOLYNOMIALS));
99
100     MSG msg;
101
102     // Main message loop:
```

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

```
153     HWND hWnd = CreateWindowW(szWindowClass, szTitle, WS_OVERLAPPEDWINDOW,
154        CW_USEDEFAULT, 0, CW_USEDEFAULT, 0, nullptr, nullptr, hInstance, nullptr);
155
156     if (!hWnd)
157     {
158        return FALSE;
159     }
160
161     ShowWindow(hWnd, nCmdShow);
162     UpdateWindow(hWnd);
163
164     return TRUE;
165  }
166  //
167  //  FUNCTION: WndProc(HWND, UINT, WPARAM, LPARAM)
168  //
169  //  PURPOSE: Processes messages for the main window.
170  //
171  //  WM_COMMAND  - process the application menu
172  //  WM_PAINT    - Paint the main window
173  //  WM_DESTROY  - post a quit message and return
174  //
175  //
176  LRESULT CALLBACK WndProc(HWND hWnd, UINT message, WPARAM wParam, LPARAM lParam)
177  {
178      switch (message)
179      {
180      case WM_COMMAND:
181          {
182              int wmId = LOWORD(wParam);
183              // Parse the menu selections:
184              switch (wmId)
185              {
186              case IDM_ABOUT:
187                  DialogBox(hInst, MAKEINTRESOURCE(IDD_ABOUTBOX), hWnd, About);
188                  break;
189              case ID_FILE_GRAPHX32773:
190                  function = 1;
191                  DialogBox(hInst, MAKEINTRESOURCE(IDD_DIALOG1), hWnd,
192                      GetCurveFittingData);
193                  DialogBox(hInst, MAKEINTRESOURCE(IDD_DIALOG2), hWnd, DrawGraph);
194                  break;
195              case ID_FILE_GRAPHX:
196                  function = 2;
197                  DialogBox(hInst, MAKEINTRESOURCE(IDD_DIALOG1), hWnd,
198                      GetCurveFittingData);
199                  DialogBox(hInst, MAKEINTRESOURCE(IDD_DIALOG2), hWnd, DrawGraph);
200                  break;
201              case ID_FILE_SLATERTYPEORBITAL1S:
202                  function = 3;
203                  DialogBox(hInst, MAKEINTRESOURCE(IDD_DIALOG1), hWnd,
204                      GetCurveFittingData);
```

```
205                 DialogBox(hInst, MAKEINTRESOURCE(IDD_DIALOG2), hWnd, DrawGraph);
206                 break;
207             case IDM_EXIT:
208                 DestroyWindow(hWnd);
209                 break;
210             default:
211                 return DefWindowProc(hWnd, message, wParam, lParam);
212             }
213         }
214         break;
215     case WM_PAINT:
216         {
217             PAINTSTRUCT ps;
218             HDC hdc = BeginPaint(hWnd, &ps);
219             // TODO: Add any drawing code that uses hdc here...
220             EndPaint(hWnd, &ps);
221         }
222         break;
223     case WM_DESTROY:
224         PostQuitMessage(0);
225         break;
226     default:
227         return DefWindowProc(hWnd, message, wParam, lParam);
228     }
229     return 0;
230 }
231
232 // Message handler for about box.
233 INT_PTR CALLBACK About(HWND hDlg, UINT message, WPARAM wParam, LPARAM lParam)
234 {
235     UNREFERENCED_PARAMETER(lParam);
236     switch (message)
237     {
238     case WM_INITDIALOG:
239         return (INT_PTR)TRUE;
240
241     case WM_COMMAND:
242         if (LOWORD(wParam) == IDOK || LOWORD(wParam) == IDCANCEL)
243         {
244             EndDialog(hDlg, LOWORD(wParam));
245             return (INT_PTR)TRUE;
246         }
247         break;
248     }
249     return (INT_PTR)FALSE;
250 }
251
252 INT_PTR CALLBACK GetCurveFittingData(
253     HWND hDlg, UINT message, WPARAM wParam, LPARAM lParam)
254 {
255     UNREFERENCED_PARAMETER(lParam);
256     switch (message)
```

```
257     {
258     case WM_INITDIALOG:
259         return (INT_PTR)TRUE;
260
261     case WM_COMMAND:
262         if (LOWORD(wParam) == IDOK || LOWORD(wParam) == IDCANCEL)
263         {
264             char buffer[128] = { };
265             GetDlgItemTextA(hDlg, IDC_EDIT1, buffer, 127);
266             x0 = atof(buffer);
267             GetDlgItemTextA(hDlg, IDC_EDIT2, buffer, 127);
268             x1 = atof(buffer);
269             npoints = GetDlgItemInt(hDlg, IDC_EDIT3, FALSE, FALSE);
270             degree = GetDlgItemInt(hDlg, IDC_EDIT4, FALSE, FALSE);
271
272             double h = (x1 - x0) / npoints, z = x0, y = 0.0;
273
274             for (int i = 1; i <= npoints; i++)
275             {
276                 if (function == 1)
277                     y = f1(z);
278                 else if (function == 2)
279                     y = f2(z);
280                 else
281                     y = f3(z);
282
283                 points0[i].x = z;
284                 points0[i].y = y;
285
286                 f[i] = y;
287                 x[i] = z;
288                 w[i] = wt(z);
289
290                 z += h;
291             }
292
293             for (int j = 1; j <= degree + 1; j++)
294                 b[j] = d[j] = s[j] = 0;
295
296             c[1] = 0;
297
298             for (int i = 1; i <= npoints; i++)
299             {
300                 d[1] += f[i] * w[i];
301                 b[1] += x[i] * w[i];
302                 s[1] += w[i];
303             }
304
305             d[1] /= s[1];
306
307             for (int i = 1; i <= npoints; i++)
308                 error[i] = f[i] - d[1];
```

```
309
310             if (degree + 1 == 1)
311                 goto Finish;
312
313             b[1] /= s[1];
314
315             for (int i = 1; i <= npoints; i++)
316             {
317                 pjm1[i] = 1.0;
318                 pj[i] = x[i] - b[1];
319             }
320
321             for (int j = 2; j <= degree + 1; j++)
322             {
323                 for (int i = 1; i <= npoints; i++)
324                 {
325                     double p = pj[i] * w[i];
326
327                     d[j] += error[i] * p;
328                     p *= pj[i];
329                     b[j] += x[i] * p;
330                     s[j] += p;
331                 }
332
333                 d[j] /= s[j];
334
335                 for (int i = 1; i <= npoints; i++)
336                     error[i] -= d[1] * pj[i];
337
338                 if (j == degree + 1)
339                     goto Finish;
340
341                 b[j] /= s[j];
342                 c[j] = s[j] / s[j - 1];
343
344                 for (int i = 1; i <= npoints; i++)
345                 {
346                     double p = pj[i];
347
348                     pj[i] = (x[i] - b[j]) * pj[i] - c[j] * pjm1[i];
349                     pjm1[i] = p;
350                 }
351             }
352
353         Finish:
354
355             for (int i = 0; i <= npoints; i++)
356             {
357                 double ortval = 0, prev = 0;
358
359                 points1[i].x = x[i];
360                 ortval = d[degree + 1];
```

```
361
362                 if (degree + 1 == 1)
363                     points1[i].y = ortval;
364                 else
365                 {
366                     for (int k = degree; k >= 1; k--)
367                     {
368                         double prev2 = prev;
369                         prev = ortval;
370                         ortval = d[k] + (x[i] - b[k]) * prev -
371                             c[k + 1] * prev2;
372                     }
373
374                     points1[i].y = ortval;
375                 }
376             }
377             EndDialog(hDlg, LOWORD(wParam));
378             return (INT_PTR)TRUE;
379         }
380         break;
381     }
382     return (INT_PTR)FALSE;
383 }
384
385 void FindMinMax(
386     double& xMin, double& xMax,
387     double& yMin, double& yMax)
388 {
389     // uses global 2D double points structure
390
391     xMin = yMin = DBL_MAX;
392     xMax = yMax = DBL_MIN;
393
394     for (int i = 1; i <= npoints; i++)
395     {
396         POINT2D pt = points0[i];
397         double x = pt.x;
398         double y = pt.y;
399
400         if (x < xMin)
401             xMin = x;
402         if (x > xMax)
403             xMax = x;
404         if (y < yMin)
405             yMin = y;
406         if (y > yMax)
407             yMax = y;
408     }
409
410     for (int i = 1; i <= npoints; i++)
411     {
412         POINT2D pt = points1[i];
```

```
413         double x = pt.x;
414         double y = pt.y;
415
416         if (x < xMin)
417             xMin = x;
418         if (x > xMax)
419             xMax = x;
420         if (y < yMin)
421             yMin = y;
422         if (y > yMax)
423             yMax = y;
424     }
425 }
426
427 void DrawFormattedText(HDC hdc, char text[], RECT rect)
428 {
429     // Draw the text with formatting options
430     DrawTextA(hdc, text, -1, &rect, DT_SINGLELINE | DT_NOCLIP);
431 }
432
433 INT_PTR CALLBACK DrawGraph(
434     HWND hDlg, UINT message, WPARAM wParam, LPARAM lParam)
435 {
436     UNREFERENCED_PARAMETER(lParam);
437     switch (message)
438     {
439     case WM_INITDIALOG:
440         return (INT_PTR)TRUE;
441
442     case WM_COMMAND:
443         if (LOWORD(wParam) == IDOK || LOWORD(wParam) == IDCANCEL)
444         {
445             EndDialog(hDlg, LOWORD(wParam));
446             return (INT_PTR)TRUE;
447         }
448         break;
449     case WM_PAINT:
450         double h = 0, pi = 0, plm = 0, theta = 0;
451         double xMax = 0, xMin = 0, yMax = 0, yMin = 0;
452         FindMinMax(xMin, xMax, yMin, yMax);
453         float xSpan = (float)(xMax - xMin);
454         float ySpan = (float)(yMax - yMin);
455         RECT rect = { };
456         GetClientRect(hDlg, &rect);
457         float width = (float)(rect.right - rect.left + 1);
458         float height = (float)(rect.bottom - rect.top - 32 + 1);
459         float sx0 = 2.0f * width / 16.0f;
460         float sx1 = 14.0f * width / 16.0f;
461         float sy0 = 2.0f * height / 16.0f;
462         float sy1 = 14.0f * height / 16.0f;
463         float deltaX = xSpan / 8.0f;
464         float deltaY = ySpan / 8.0f;
```

```
465         float xSlope = (sx1 - sx0) / xSpan;
466         float xInter = (float)(sx0 - xSlope * xMin);
467         float ySlope = (sy0 - sy1) / ySpan;
468         float yInter = (float)(sy0 - ySlope * yMax);
469         float px = 0, py = 0, sx = 0, sy = 0;
470         PAINTSTRUCT ps;
471         POINT wPt = { };
472         HDC hdc = BeginPaint(hDlg, &ps);
473         int i = 0;
474         float x = (float)xMin;
475         float y = (float)yMax;
476         px = x;
477         py = y;
478         sx = xSlope * px + xInter;
479         sy = ySlope * py + yInter;
480         MoveToEx(hdc, (int)sx, (int)sy0, &wPt);
481         char buffer[128] = { };
482
483         while (i <= 8)
484         {
485             sx = xSlope * x + xInter;
486             MoveToEx(hdc, (int)sx, (int)sy0, &wPt);
487             LineTo(hdc, (int)sx, (int)sy1);
488             sprintf_s(buffer, "%5.2f", x);
489             SIZE size = { };
490             GetTextExtentPoint32A(
491                 hdc,
492                 buffer,
493                 strlen(buffer),
494                 &size);
495             RECT textRect = { };
496             textRect.left = (long)(sx - size.cx / 2.0f);
497             textRect.right = (long)(sx + size.cx / 2.0f);
498             textRect.top = (long)sy1;
499             textRect.bottom = (long)(sy1 + size.cy / 2.0f);
500             DrawFormattedText(hdc, buffer, textRect);
501             x += deltaX;
502             i++;
503         }
504
505         i = 0;
506         y = (float)yMin;
507
508         while (i <= 8)
509         {
510             sy = ySlope * y + yInter;
511             MoveToEx(hdc, (int)sx0, (int)sy, &wPt);
512             LineTo(hdc, (int)sx, (int)sy);
513
514             if (i != 0)
515             {
516                 sprintf_s(buffer, "%5.2f", y);
```

```
517                 SIZE size = { };
518                 GetTextExtentPoint32A(
519                     hdc,
520                     buffer,
521                     strlen(buffer),
522                     &size);
523                 RECT textRect = { };
524                 textRect.left = (long)(sx0 - size.cx - size.cx / 5.0f);
525                 textRect.right = (long)(sx0 - size.cx / 2.0f);
526                 textRect.top = (long)(sy - size.cy / 2.0f);
527                 textRect.bottom = (long)(sy + size.cy / 2.0f);
528                 DrawFormattedText(hdc, buffer, textRect);
529             }
530 
531             y += deltaY;
532             i++;
533         }
534 
535         px = (float)points0[1].x;
536         py = (float)points0[1].y;
537         sx = xSlope * px + xInter;
538         sy = ySlope * py + yInter;
539         MoveToEx(hdc, (int)sx, (int)sy, &wPt);
540 
541         HGDIOBJ rPenNew = NULL;
542         HGDIOBJ hPenOld = NULL;
543         rPenNew = CreatePen(PS_SOLID, 2, RGB(255, 0, 0));
544         hPenOld = SelectObject(hdc, rPenNew);
545 
546         for (int j = 2; j <= npoints; j++)
547         {
548             px = (float)points0[j].x;
549             py = (float)points0[j].y;
550             sx = xSlope * px + xInter;
551             sy = ySlope * py + yInter;
552             LineTo(hdc, (int)sx, (int)sy);
553         }
554 
555         SelectObject(hdc, hPenOld);
556         DeleteObject(rPenNew);
557 
558         px = (float)points1[1].x;
559         py = (float)points1[1].y;
560         sx = xSlope * px + xInter;
561         sy = ySlope * py + yInter;
562         MoveToEx(hdc, (int)sx, (int)sy, &wPt);
563 
564         HGDIOBJ bPenNew = CreatePen(PS_SOLID, 2, RGB(0, 0, 255));
565         hPenOld = SelectObject(hdc, bPenNew);
566 
567         for (int j = 2; j <= npoints; j++)
568         {
```

```
569             px = (float)points1[j].x;
570             py = (float)points1[j].y;
571             sx = xSlope * px + xInter;
572             sy = ySlope * py + yInter;
573             LineTo(hdc, (int)sx, (int)sy);
574         }
575
576         SelectObject(hdc, hPenOld);
577         DeleteObject(bPenNew);
578         EndPaint(hDlg, &ps);
579         return (INT_PTR)FALSE;
580         break;
581     }
582
583     return (INT_PTR)FALSE;
584 }
```