수치 해석학 기말 프로젝트

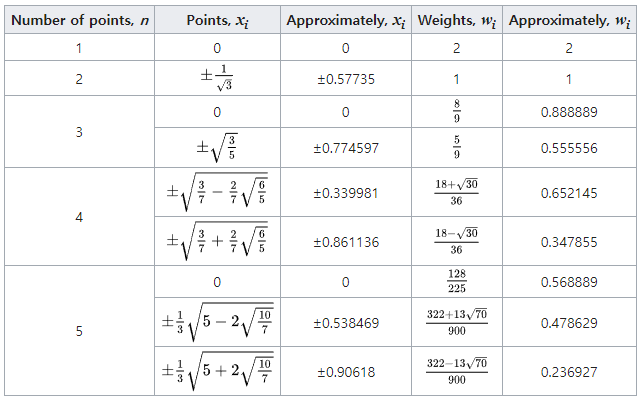
20173596 최윤정

1. Code

Each problem in the Final project is solved by FinalProject1, FinalProject2, FinalProject3, FinalProject4\_5\_6\_7. I will give more detail about code in each problem section.

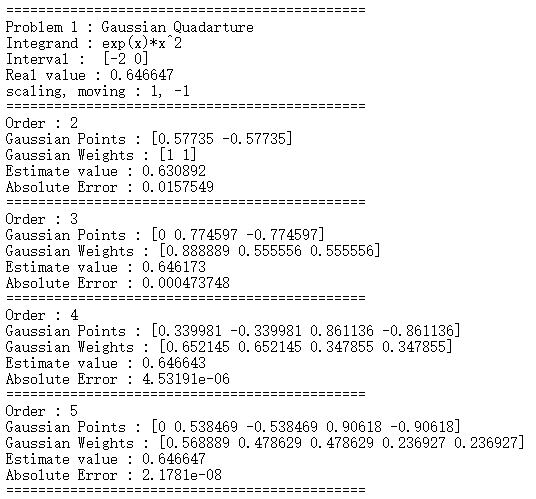
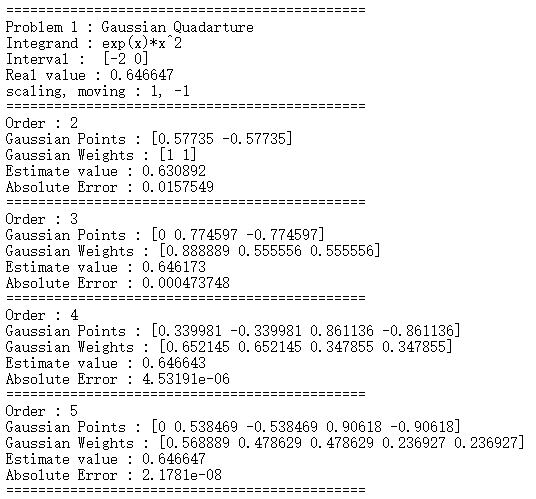
1. Problem 1 : Gaussian Quadrature

By Wikipedia, the Gaussian Quadrature with n points (n=1, 2, 3, 4, 5) are given as follow:



Thus I define a function **void GaussianQuadrature(FVector<double>& points, FVector<double>& weights, int num\_points)** which updates **points** to Gaussian points and **weights** to Gaussian weights corresponding to points. Since the interval of the integration is [-2, 0] which is not [-1, 1], the scaling and moving are necessary when I sum the values of the function.

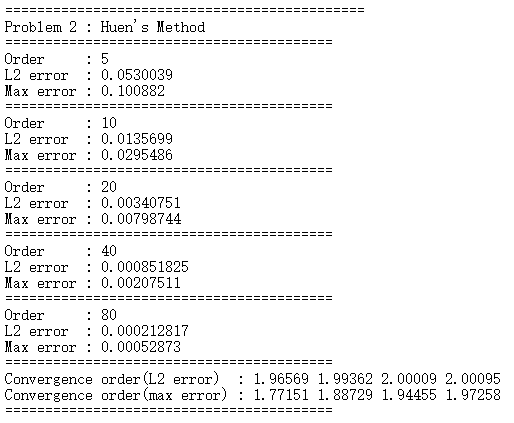
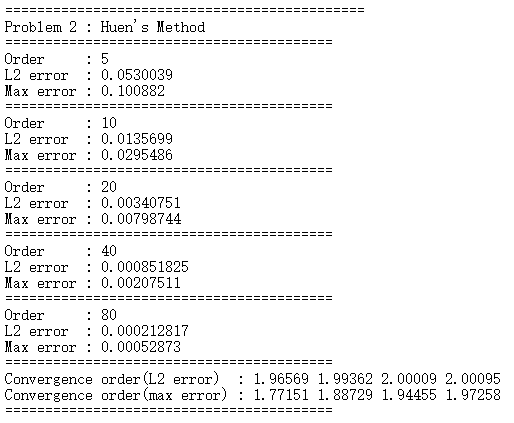
The result is as follow:



As the order increases, the absolute error between the estimated value and real value decreases.

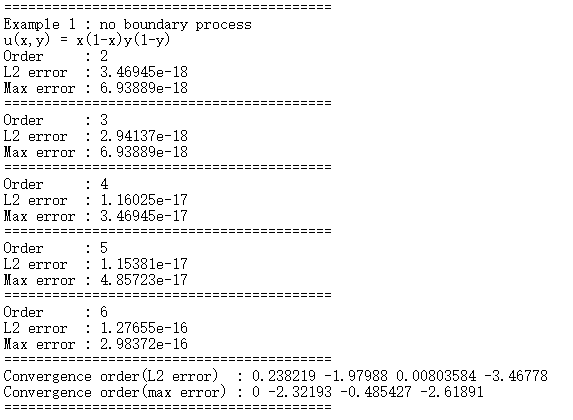
1. Problem 2 : Solve the O.D.E by Heun’s method.

First, the given ordinary differential equation is a first order linear equation, the exact solution can be obtained easily. The exact solution is . I tested the Huen’s method with orders n=5, 10 ,20 ,40, 80 to obtain convergence orders for and . The result is as follow:



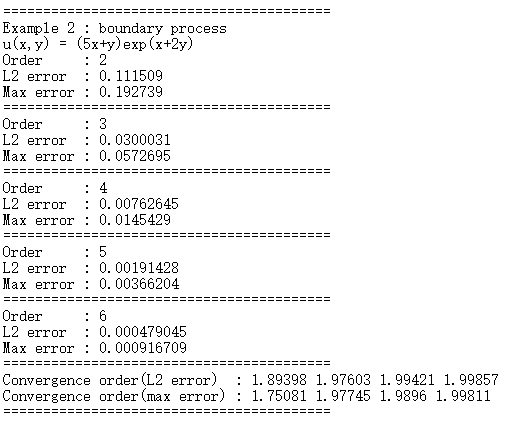
The convergence orders of both norms converges to 2 and the Huen’s method is a second oreder method.

1. Problem 3 : Finite Difference Method

I defined a function **void make\_Stiffeness(FMatrix<double> &mA, int order, bool h\_term = false)** which makes the input matrix mA a (order\*order)\*(order\*order) matrix obtained by f.d.m. If the input **h\_term** is false, the matrix mA is not multiplied by h=1/(order+1). This function is used in Problem4 ~ Problem7. When I solved the linear system obtained by f.d.m scheme, I use Conjugate Gradient Method. This solver is defined in the class **Solver**. I solve two functions. One is which has zero boundary. The other is . I tested the code with the levels 2,3,4,5,6 to to obtain convergence orders for and . The result of is as follow: 

For , since the error between the estimated values and real values is too small, the convergence order is meaningless. However, the and of the errors shows that the estimated solutions obtained by f.d.m are correct.

The result of is as follow:

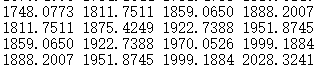


Since is nonzero on the boundary, the boundary process is necessary. The convergence order of both and converges to 2.

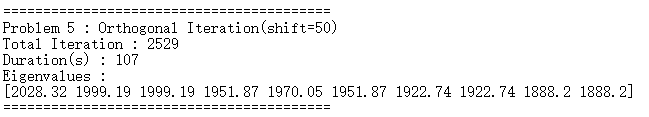
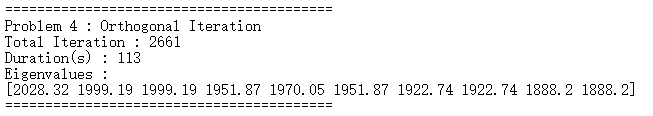
1. Problem 4 & 5 : Orthogonal Iteration

For orthogonal iteration, I use the Modified Gram-Schmidt(MGS) to decompose the matrix X to QR. I defined a class **QR\_Decomposition** which have three functions for QR decomposition : Classic Gram-Schmidt(CGS), Modified Gram-Schmidt(MGS) and Householder. In these two problems, I used the function **void MGS(FMatrix<double>&A, FMatrix<double>& Q, FMatrix<double>& R)** which makes the matrix Q and R reduced forms of Q and R for the matrix A. I report the execution time and the number of iterations of the orthogonal iteration. As the orthogonal iteration repeated, the diagonal entries of R converges to the 10 largest eigenvalues. Thus I set the stopping criteria for the orthogonal iteration as the between the diagonal entries of the current R and those of the previous R. Thus if this is smaller than epsilon=1e-10, the iteration is terminated.

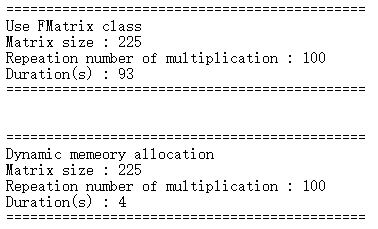
Note that the 10 largest eigenvalues of the matrix are given as follow:



The result of the orthogonal iteration with/without shift(=50) is as follow:



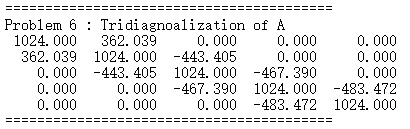
Both iteration methods give us the correct 10 eigenvalues. By comparing the number of iterations, the shift makes the convergence of the orthogonal iteration somewhat fast. However, in both case, the execution time is slow comparing with the MATLAB code. I have checked that the computation time for matrix multiplication (in this case, multiplication of a 225\*10 matrix and a 10\*10 matrix) dominates the computation time for QR decomposition. Moreover, I made a class for Matrix which is used in the code. I tested the computation time for matrix multiplication using FMatrix class and using just dynamic memory allocation without FMatrix. The result is as follow:



The multiplication not using FMatrix class is much faster than the multiplication using FMatrix. I think that the FMatrix class that I defined does not have good performance and so the execution time for the orthogonal iteration is long.

1. Problem 6 : Tri-diagonalization by Householder Transformation

For tri-diagonalization, I defined a class **HouseholderTransformation**. This class has two main functions. One is **void Tridiagonalization(FMatrix<double>& mA)** for tri-diagonalization of a symmetric matrix. The other is **void UpperHessenberg(FMatrix<double>& mA)** for upper-Hessenberg form of a matrix. I use void Tridiagonalization(FMatrix<double>& mA) to get a tri-diagonal matrix from the matrix A. This matrix is used again in Problem7. The result is as follow:

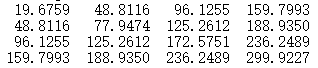


1. Problem 7 : QR Iteration

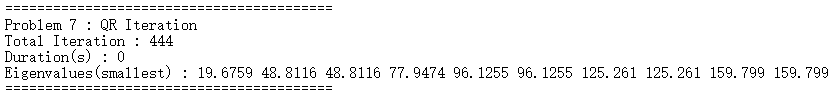
For QR iteration, I used Householder transformation to obtain QR decomposition. Since the size of matrix is somewhat large, if we apply QR iteration directly, the execution time is very long. However, the matrix A becomes a symmetric tri-diagonal matrix by Problem 6. Thus, the matrix is sparse and so the algorithm can be improved. Moreover, it is known that the upper Hessenberg form is preserved under QR iteration. If where A is symmetric, then the next A is is also symmetric. Thus the tri-diagonal property is preserved under QR iteration. When the Householder transform is applied to get the upper triangular matrix R, since A is tri-diagonal, the vector w corresponding to each Householder transform has exactly two nonzero sequential entries. With this fact, the computation of R is reduced to O(n). When is computed, since R has off-diagonal nonzero entries as (k,k+1) and (k,k+2) and the vector for each Householder transform has only two nonzero entries, the computation of is also reduced. Finally, I used the deflation: if the stopping criteria (defined later) is satisfied, QR iteration is applied to a submatrix which is left-up part of the diagonal entry. Since converges to a diagonal matrix repeating QR iteration, the diagonal entries of converge to all eigenvalues. I set the stopping criteria as **abs(A[num][num - 1]) < epsil\*A[num][num]** where epsil=1e-16. Here, the QR iteration is applied to a submatrix A[0:num][0:num]. Thus if the off-diagonal part A[num][num-1] is sufficiently small w.r.t the diagonal entry A[num][num] then the submatrix A[0:num][0:num] can be considered as a block upper triangular matrix. Thus A[num][num] is an eigenvalue of the original A. Then the QR iteration is applied to a submatrix A[0:num-1][0:num-1]. Repeat this procedure until num is zero then all eigenvalues are obtained.

Moreover I used the right-down entry of each submatrix as the shift = A[num][num].

Note that the 10 largest eigenvalues of the matrix are given as follow:

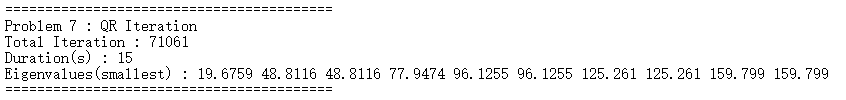


The result is as follow:



The result shows that the QR iteration provides correct eigenvalues. Moreover, since I improved the algorithm using the tri-diagonal property, the execution time is much faster than that of the orthogonal iteration in Problem 5 and 6 even though the orthogonal iteration finds only 10 eigenvalues when the QR iteration find all 225 eigenvalues.

Finally, I tested the QR iteration without any shift so that I can check the usefulness of shift.



Although the QR iteration without any shift also finds all eigenvalues, the number of iteration for QR iteration is much larger than that of QR iteration with appropriate shift.