

Appendix A Calculation of Coefficients

The orthogonal collocation method differs from the ordinary collocation method by using collocation points that are the roots of certain Jacobi orthogonal polynomials. These roots are selected because they form the basis of highly accurate numerical integration or quadrature methods. To apply orthogonal collocation, we first need the polynomial roots or quadrature base points, \mathbf{x} . In addition to the base points, the quadrature weights, \mathbf{W} , and the various matrices \mathbf{A} , \mathbf{B} , etc. are also needed. The weights, \mathbf{W} , give highly accurate approximations of integrals and the matrix operators, \mathbf{A} , \mathbf{B} , etc. give approximations of derivatives.

The roots of the Jacobi polynomials must be determined first. Once the roots are known, the calculation of the quadrature weights and derivative operators is relatively straight forward. For our purposes, we are interested in polynomials that are orthogonal on the interval 0 to 1, with respect to a general weight function:

$$\omega(x) = (1 - x)^\alpha x^\beta \tag{A-1}$$

In Eq. (A-1), x represents x^2 for the symmetric cases. For the geometry, symmetry and quadrature methods considered here, the values of α and β are listed in Table A-1. The seemingly strange values of β for the symmetric cases in Table A-1 result because the quadrature formulae are based on polynomials in x^2 . For the symmetric cases, the Lobatto quadrature is more correctly a Radau quadrature, since only one endpoint is used due to symmetry.

Table A-1 Weight Exponents for Jacobi Polynomials

	Planar	Cylindrical	Spherical
Nonsymmetric, Gauss	$\alpha = 0, \beta = 0$	n.a.	n.a.
Nonsymmetric, Lobatto	$\alpha = 1, \beta = 1$	n.a.	n.a.
Symmetric, Gauss	$\alpha = 0, \beta = -1/2$	$\alpha = 0, \beta = 0$	$\alpha = 0, \beta = 1/2$
Symmetric, Lobatto	$\alpha = 1, \beta = -1/2$	$\alpha = 1, \beta = 0$	$\alpha = 1, \beta = 1/2$

Stroud and Secrest [1] have tabulated the roots and quadrature weights to high accuracy for numerous cases through large n . Rather than using tabular values, it is more desirable to have a computer code which will produce the desired quantities when the specific Jacobi polynomial and the number of points is specified. Hildebrand [2] provides recursive formulae for calculating the coefficients of the Jacobi polynomials. The roots of the polynomials can be found by standard iterative methods. Burkardt [3] and Villadsen and Michelsen [4] give codes for calculating the polynomial roots by Newton-Raphson iteration. Quadrature weights are also calculated. These codes have been found to work well, but they are not written in an object oriented style. Burkardt provides Fortran 90, C++ and Matlab versions of his code, which is based on the original FORTRAN 77 code of Stroud and Secrest. Golub and Welsch [5] present an alternate method based on calculating eigenvalues of a tridiagonal matrix. GAUSSQ [6] is an available FORTRAN 77 code which uses the eigenvalue method to determine the roots and quadrature weights.

Our code [7] uses the Newton-Raphson method to determine the roots. Then the quadrature weights, interpolating polynomials and derivative operators are calculated using the methods described below. This code provides a toolkit for solving problems with not only Orthogonal Collocation but also other Methods of Weighted residuals. The code is implemented in an object oriented style as a Fortran 90 module and a C++ class. Currently, the C++ code uses dynamic link libraries rather than native code. It would be relatively easy to use Burkardt's root calculation code to implement a native C++ class, without altering the current programming interface.

Once the roots have been found, the quadrature weights and derivative approximations can be calculated. First, we note that the Lagrange interpolating polynomials can be represented in their fundamental form [8]:

$$l_i(x) = \prod_{\substack{j=0 \\ j \neq i}}^{n+1} \frac{(x - x_j)}{(x_i - x_j)} = \frac{p(x)}{(x - x_i)p'(x_i)} \quad (\text{A-2})$$

where:

$$p(x) = \prod_{j=0}^{n+1} (x - x_j) \quad \text{and} \quad p'(x_i) = \left. \frac{dp(x)}{dx} \right|_{x_i} = \prod_{\substack{j=0 \\ j \neq i}}^{n+1} (x_i - x_j) \quad (\text{A-3})$$

The n^{th} Jacobi polynomial is related to $p(x)$ by:

$$p(x) = x(1-x)P_n(x) / A_n \quad (\text{A-4})$$

Where P_n is the orthogonal polynomial and A_n is its leading coefficient. The same representation can be used for the symmetric case by replacing x by x^2 . By using various recursion and derivative relationships [2,8], there are many different ways to represent the formulae for the quadrature weights. We prefer the following ones which depend only on the polynomial $p(x)$ and not higher or lower ones in the series of orthogonal polynomials. Since Eqs. (A-2) and (A-3) include the endpoints in the continued product, the formulae below appear to be slightly different from those given elsewhere [2,9].

We list the specific quadrature formulae of interest here, for the nonsymmetric case:

$$W_i \propto \frac{x_i(1-x_i)}{p'(x_i)^2} \quad \text{for Gauss, and} \quad (\text{A-4})$$

$$W_i \propto \frac{1}{p'(x_i)^2} \quad \text{for Lobatto}$$

For the symmetric case:

$$W_i \propto \frac{(1-x_i^2)}{x_i^2 p'(x_i^2)^2} \quad \text{for Gauss, and} \quad (\text{A-4})$$

$$W_i \propto \frac{1}{x_i^2 p'(x_i^2)^2} \quad \text{for Lobatto}$$

Formulae for the proportionality constants are available [2,9], but we find it simpler to determine them such that $\sum W_i = 1/(\gamma + 1)$, where $\gamma = 0, 1, 2$ for planar, cylindrical, spherical geometry.

The approximation for the first derivative can be determined by direct differentiation of Eq. (A-2). For the nonsymmetric case:

$$A_{ij} = \frac{p'(x_i)}{(x_i - x_j)p'(x_j)} \quad \text{for } i \neq j, \text{ and}$$

$$A_{ii} = \sum_{\substack{j=0 \\ j \neq i}}^{n+1} \frac{1}{(x_i - x_j)} \quad (\text{A-7})$$

and for the symmetric case:

$$A_{ij} = \frac{2x_i p'(x_i^2)}{(x_i^2 - x_j^2)p'(x_j^2)} \quad \text{for } i \neq j, \text{ and}$$

$$A_{ii} = \sum_{\substack{j=0 \\ j \neq i}}^{n+1} \frac{2x_i}{(x_i^2 - x_j^2)} \quad (\text{A-8})$$

Alternatively, the diagonal elements can be calculated so the row sums are zero.

The second derivative can be calculated from the first derivative approximation. For the nonsymmetric case:

$$B_{ij} = \sum_{k=0}^{n+1} A_{ik} A_{kj} \quad (\text{A-9})$$

For the symmetric case, the Laplacian operator is not as simple as Eq. (A-9). Since the interpolating polynomial is even, its first derivative is odd. The first derivative operator from Eq. (A-8) is only valid if it operates on a symmetric quantity. Instead, a similar operator is needed for odd functions. The Lagrange interpolating polynomial for an odd function is related to the even function interpolating polynomial by:

$$\widehat{\ell}_i = \frac{x}{x_i} \prod_{\substack{j=1 \\ j \neq i}}^{n+1} \frac{x^2 - x_j^2}{x_i^2 - x_j^2} = \frac{x}{x_i} \ell_i(x^2) \quad (\text{A-10})$$

The operator needed is then given by:

$$A_{ij}^* = \frac{1}{x^\gamma} \frac{d}{dx} \left[x^\gamma \widehat{\ell}_j \right]_{x_i} = \frac{\gamma+1}{x_i} \delta_{ij} + \frac{x_i}{x_j} A_{ij} \quad (\text{A-11})$$

The Laplacian operator can now be calculated by $\mathbf{B} = \mathbf{A}^* \mathbf{A}$.

The Laplacian operator, \mathbf{B} , can also be determined by direct differentiation of Eq. (A-2). For the nonsymmetric case, the values from direct differentiation are:

$$B_{ij} = \frac{2p'(x_i)}{(x_i - x_j)p'(x_j)} \left[\sum_{\substack{k=0 \\ k \neq i \\ k \neq j}}^{n+1} \frac{1}{(x_i - x_k)} \right] = 2A_{ij} \left[A_{ii} - \frac{1}{(x_i - x_j)} \right] \quad \text{for } i \neq j, \text{ and}$$

$$B_{ii} = \sum_{\substack{j=0 \\ j \neq i}}^{n+1} \left[\frac{1}{x_i - x_j} \sum_{\substack{k=0 \\ k \neq i \\ k \neq j}}^{n+1} \frac{1}{(x_i - x_k)} \right] = A_{ii}^2 - \sum_{\substack{j=0 \\ j \neq i}}^{n+1} \frac{1}{(x_i - x_j)^2} \quad (\text{A-12})$$

Similar relationships can be developed for the symmetric case.

In finite element terminology the **C** array is often called the stiffness matrix. It produces the symmetric positive definite representation of the Laplacian, see Eq. (23) in the main text. It is easily calculated from the other arrays. For Lobatto points or symmetric problem with Gauss points:

$$C_{ji} = \sum_{k=0}^{n+1} W_k A_{kj} A_{ki}$$

For nonsymmetric problems with Gauss points, the above equation cannot be used because the quadrature is not accurate enough to integrate the diffusion term. However, if Eq. (24) of the main text is used to calculate **C**, the Laplacian representation will be symmetric and the approximations will be equal to the normal representation multiplied by $-\mathbf{W}$ at the interior points.

The matrix **M** in Eqs. (20) or (25) (of the main text) is often called the mass matrix. It is diagonal and equal to the quadrature weights for symmetric problems, since the integration is exact. For nonsymmetric problems it can be calculated by using a quadrature formula with one additional base point.

We have developed a Fortran 90 module for calculating:

1. Base points, **x**
2. Quadrature weights, **W**
3. Lagrange interpolating polynomials, $\ell(x)$
4. First derivative operators, **A** and **A***
5. Laplacian operator, **B**
6. Symmetric Laplacian or stiffness matrix, **C**
7. Mass matrix for Galerkin or moments methods, **M**, Eq.(20) or (25)

These are implemented as array valued functions which return the requested array. The code has also been used to create dynamic link libraries for use with Excel or C++. The code as well as various demonstration programs (Fortran 90, C++, and Excel) may be obtained at reference [7].

The calculations are implemented so the precision can be changed by a simple parameter redefinition (parameter *float*). Testing has been performed with double and quad precision (8 and 16 byte reals), with *n* as high as 100. The tests found that with double precision (8 byte reals) the values are valid to at least 14 or 15 decimal places. For example, the quadrature formulae have been tested by integrating x^k . When *k* is small enough to give exact integration, the fractional error is less than 10^{-15} and 10^{-33} for double and quad precision respectively. Interestingly, when the power *k* is one larger than that necessary for exact integration, i.e. $k = 2n$ for Gaussian quadrature or $k = 2n+2$ for Lobatto quadrature, the fractional integration error is $< 10^{-16}$ when $k > 28$, i.e. $n > 13$ or 14. For larger *n*, the integration errors are the same order of magnitude as roundoff errors for double precision. Apparently, these roundoff errors are not manifested in the matrix operators. We have done convergence analysis in double precision with up to $n = 100$ with no indication of roundoff in the convergence trend.

References

1. Stroud and Secrest, *Gaussian Quadrature Formulas*, Prentice-Hall, Englewood Cliffs, NJ (1966).
2. Hildebrand, F.B., *Introduction to Numerical Analysis*, 2nd ed., Dover Publications, New York, NY (1987).
3. Burkardt, J., Quadrule - a library which defines quadrature rules for approximating integrals, available in Fortran 90 and C++, people.sc.fsu.edu/~jburkardt/
4. Villadsen, J.V. and Michelsen, M.L., *Solution of Differential Equation Models by Polynomial Approximation*, Prentice-Hall, Englewood Cliffs, NJ (1978)
5. Golub, G.H. and Welsch, J.H., "Calculation of Gaussian Quadrature Rules", *Mathematics of Computation*, vol. 23 (April 1969), pp. 221-230.
6. GAUSSQ, National Institute of Standards and Technology, Guide to Available Mathematical Software – gams.nist.gov/serve.cgi/Module/GO/GAUSSQ/8182
7. Young, L.C., *Orthogonal Collocation Revisited*, tildentechnologies.com/Numerics.
8. Szegő, G., *Orthogonal Polynomials*, 4th ed., American Mathematical Soc., Providence, RI (1975).
9. Krylov, V.I., *Approximate Calculation of Integrals*, Macmillan, New York, NY (1962).