Linear Algorithm for Gaussian Quadrature

Jeffery Hein
Dartmouth College
MATH 126

March 9, 2012

1. Introduction

Roots of the Legendre polynomials are precisely the nodes used in Gaussian quadrature. Finding such roots in an efficient manner can greatly increase the speed of numerical integration. In this project, we explore the Glaser-Liu-Rokhlin algorithm [2] which yields the roots of Legendre polynomials in linear time. Previous methods include the Golub-Welsch method, which solves for the eigenvalues of a tri-diagonal matrix. Known algorithms can find these eigenvalues in quadratic time, and so for a large number of nodes, this method becomes cumbersome. For a small number of nodes, the Golub-Welsch method is preferable, since quadratic algorithms tend to be tractable for smaller inputs. However, for a large number of nodes, a linear time algorithm becomes an invaluable tool!

2. Prüfer Transformation

In this paper we will be concerned primarily with solutions to second-order linear differential equations of the form

$$p(x)u''(x) + q(x)u'(x) + s(x)u(x) = 0, \qquad a < x < b,$$

where p, q, s are smooth on the interval of definition. More specifically, we are interested in *self-adjoint* equations, which are classified by the property that $p' = q$. With this in mind, these equations can then be written as

$$\frac{d}{dx}\left[p(x)\frac{du}{dx}\right] + s(x)u(x) = 0, \qquad \text{for all } a < x < b. \tag{1.1}$$

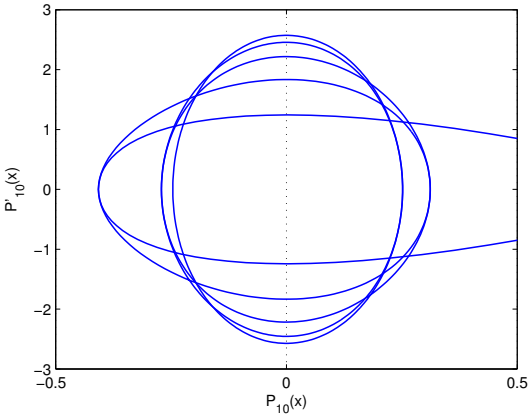
Our interest stems from our desire to find the roots of Legendre polynomials, a family of polynomials which are the solution to a family of self-adjoint equations. We denote the n-th Legendre polynomial by $P_n(x)$ and note that they are defined as the solution to the following self-adjoint equation

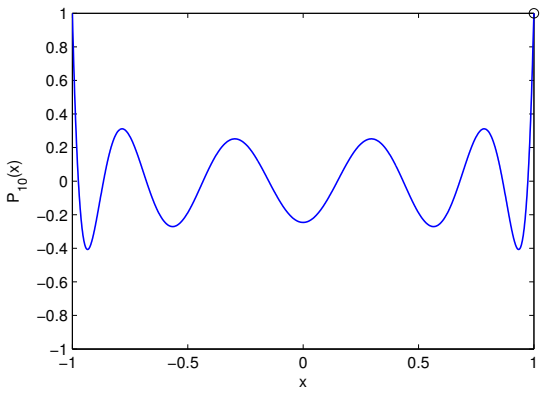
$$\frac{d}{dx}\left[(1-x^2)\frac{d}{dx}P_n(x)\right] + n(n+1)P_n(x) = 0, \qquad \text{for all } -1 < x < 1. \tag{1.2}$$

2.1 Poincaré Phase Plane

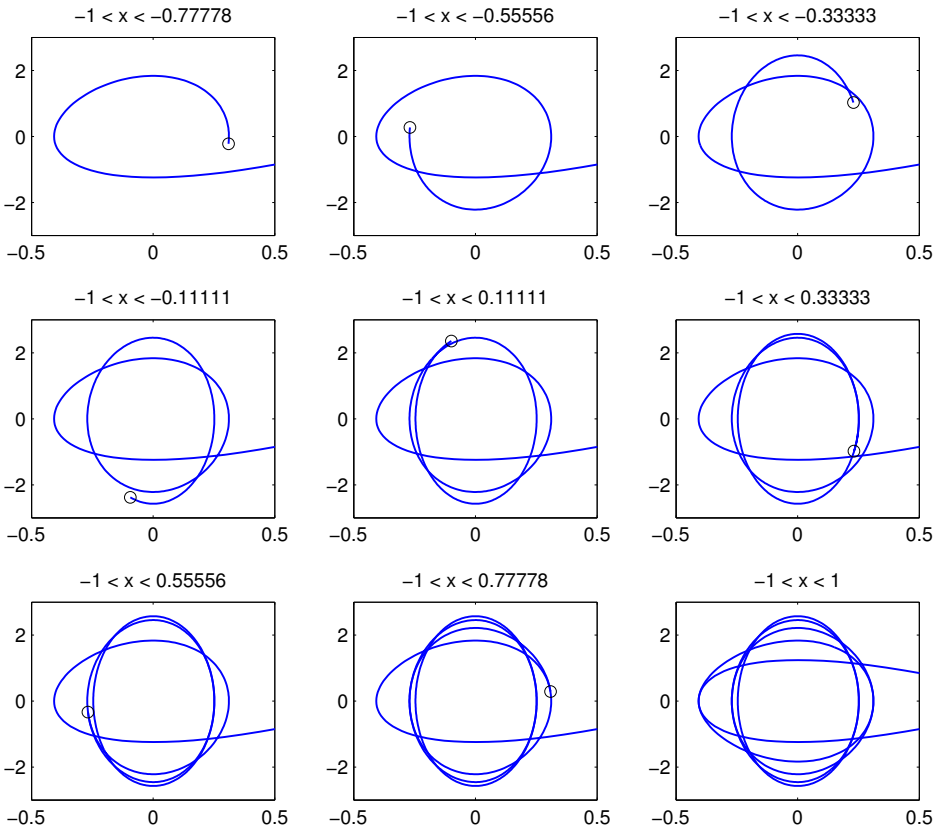
For now, we will forget about the Legendre polynomials, focusing on general self-adjoint equations. Precisely, our goal will be find the roots of the solutions of self-adjoint equations. In our pursuit, we will make the assumption that $p(x) > 0$ for all $a < x < b$. Given u, a solution to a self-adjoint equation, we wish to capture exactly how u oscillates between its roots. To determine such behavior, we will look at the relationship between the solution u and its derivative u'. Formalizing this, we

will look at how it behaves in the (u, u')-phase plane, called the Poincaré phase plane.





It is evident from the very definition of our phase plane that a root of u occurs precisely when the curve passes through the u'-axis in the phase plane. It's clear that $u' > 0$ when it crosses above the upper-half plane, and $u' < 0$ when it crosses in the lower-half plane. Since we are dealing with second-order differential equations, then $u(c) = u'(c) = 0$ implies that $u \equiv 0$, and so we never pass through the origin! The next sequences of figures traces the evolution of the phase plane curve of $P_{10}(x)$



and highlights the polar nature of how the roots and critical points of u evolve as we move along our interval of definition. Our goal will be to capture this behavior by making substitutions allowing us to extract the *amplitude* and the *phase* in this plane.

2.2 The Transformation

To this end, we make the following substitution, called the *Prüfer Transformation* [1],

$$p(x)u'(x) = r(x)\sin\theta(x)\gamma(x) \qquad \text{and} \qquad u(x) = r(x)\cos\theta(x), \tag{1.3}$$

where $\gamma : (a, b) \to \mathbb{R}^+$ is an arbitrary positive differentiable function, and θ and r are called the phase and amplitude variables, respectively. The inclusion of the γ term may seem a bit random, but it will effectively allow us to improve the quality of our numerical results as we'll later see. Also, it is important to note that we are actually "reversing" our phase orientation by treating θ as a clockwise phase.

Solving for $r(x)$ and $\theta(x)$ in terms of our known functions, we obtain

$$r^2 = \left(\frac{pu'}{\gamma}\right)^2 + u^2 \qquad \text{and} \qquad \theta = \arctan\left(\frac{pu'}{\gamma u}\right). \tag{1.4}$$

We first observe that since p and γ are both nonzero on our interval, then $r(c) = 0$ if and only if $u(c) = 0$ and $u'(c) = 0$; this can only occur if $u \equiv 0$, and so r is nonzero in our interval. Looking back at our subsitutions in (1.3), it is now evident that $u(c) = 0$ if and only if $\cos(\theta(c)) = 0$, and $u'(c) = 0$ if and only if $\sin(\theta(c)) = 0$.

The impact of this observation is that the roots of $u(x)$, in our interval, occur precisely at the points where $\theta(x) = \pm\frac{\pi}{2}$, which is consistent with our observations in in the Poincaré phase plane. Likewise, we know that we have reached a critical point, i.e. $u'(x) = 0$ in our interval, when $\theta(x) = 0$. Unfortunately, the range of $\theta(x)$ limits our ability to find roots of u; since $\theta \to \pm\frac{\pi}{2}$ only as $pu'/\gamma u \to \pm\infty$, this form isn't terribly helpful. To remedy this, we will express x as a function of θ; more specifically, we will use what we already know from (1.4) to define a differential equation with $x(\theta)$ as its unique solution.

3. Deriving A Helpful ODE

To do this, we first observe that $\tan\theta = pu'/\gamma u$ and differentiate, obtaining

$$\begin{aligned}
\sec^2\theta\cdot\theta' &= \frac{(pu')'}{\gamma u} - \frac{(\gamma u)'pu'}{(\gamma u)^2} \\
&= -\frac{su}{\gamma u} - \frac{(\gamma u)'pu'}{(\gamma u)^2} \qquad \text{(Equation (1.1))} \\
&= -\frac{s}{\gamma} - \frac{(\gamma u)'}{\gamma u}\tan\theta \qquad \text{(Equation (1.4))} \\
&= -\frac{s}{\gamma} - \left(\frac{\gamma u' + \gamma' u}{\gamma u}\right)\tan\theta \\
&= -\frac{s}{\gamma} - \left(\frac{u'p}{up} + \frac{\gamma'}{\gamma}\right)\tan\theta \\
&= -\frac{s}{\gamma} - \left(\frac{r\gamma\sin\theta}{rp\cos\theta} + \frac{\gamma'}{\gamma}\right)\tan\theta \qquad \text{(Equation (1.3))} \\
&= -\frac{s}{\gamma} - \left(\frac{\gamma}{p}\tan\theta + \frac{\gamma'}{\gamma}\right)\tan\theta \\
&= -\frac{s}{\gamma} - \frac{\gamma}{p}\tan^2\theta - \frac{\gamma'}{\gamma}\tan\theta.
\end{aligned}$$

Multiplying both sides by $\cos^2\theta$ and simplifying with the identity $\sin(2\theta) = 2\sin\theta\cos\theta$, we end up with the differential equation

$$\frac{d\theta}{dx} = -\frac{s}{\gamma}\cos^2\theta - \frac{\gamma}{p}\sin^2\theta - \frac{\gamma'}{\gamma}\cdot\frac{\sin(2\theta)}{2} \tag{1.5}$$

The form of (1.5) is now very telling as for reason for including γ in our initial substitution. Up to this point, we have been able to develop everything for an arbitrary γ; however, now we have s and $1/p$, which can scale fairly dramatically. Such differences between s and $1/p$ can cause severe problems when we attempt to numerically solve such an ODE, and so we need to choose γ in a way that acts to balance s and $1/p$. Choosing

$$\gamma = \sqrt{sp}$$

remedies this problem, as it strikes a balance between the two terms (this comes from solving $s/\gamma = \gamma/p$ for γ). This tells us that $\gamma' = (sp)'/2\gamma$, and so rewriting (1.5), we have

$$\frac{d\theta}{dx} = -\sqrt{\frac{s}{p}} - \left(\frac{p'}{p} + \frac{s'}{s}\right)\cdot\frac{\sin(2\theta)}{4} \tag{1.6}$$

Ignoring the pathologies where $d\theta/dx = 0$, i.e. our function stalls in the Poincaré phase plane, we can treat x as a function with θ as a variable. This produces the differential equation

$$\frac{dx}{d\theta} = -\left(\sqrt{\frac{s}{p}} + \left(\frac{p'}{p} + \frac{s'}{s}\right)\cdot\frac{\sin(2\theta)}{4}\right)^{-1}, \tag{1.7}$$

which we seek to solve numerically. First, however, assume that $x(\theta)$ is a solution to this equation, and so we easily observe that $x(\theta + n\pi)$ is also a solution to (1.7), since $\sin(2(\theta + n\pi)) = \sin(2\theta + 2n\pi) = \sin(2\theta)$ for all θ and all integers n.

3.1 Finding Roots

The importance of (1.7) is that we can now start with a root of u in the interval (a, b) and then solve an initial-value problem to find another root. Moreover, we don't just find an arbitrary root, we find the *next* root of u in the interval. Essential this means that if $\tilde{x}_0 \in (a, b)$ is a root of u, i.e. $u(\tilde{x}_0) = 0$, then solving the initial-value problem

$$\begin{cases} \dfrac{dx}{d\theta} = -\left(\sqrt{\dfrac{s}{p}} + \left(\dfrac{p'}{p} + \dfrac{s'}{s}\right) \cdot \dfrac{\sin(2\theta)}{4}\right)^{-1}, \quad -\pi/2 < x < \pi/2 \\ \\ x(\pi/2) = \tilde{x}_0 \end{cases} \tag{1.8}$$

gives us a means of approximating the next successive root $\tilde{x}_1$. Solving this IVP with Runge-Kutta in "reverse" from $\pi/2$ to $-\pi/2$ (recall that our phase orientation is clockwise), we yield an approximation to the next largest root of u, i.e. $\tilde{x}_1 := x(-\pi/2)$, with $\tilde{x}_1 > \tilde{x}_0$ such that there are no roots of u between them. In general, this root is only going to be valid to about 3-4 significant digits, and so we would naturally like to improve this approximation. As an approximate root of u, $\tilde{x}_1$ is a perfect candidate for the starting point of the iterative Newton's Method. We will later see that for Legendre polynomials, this method can be replaced altogether with a simpler, faster method of approximating roots.

3.2 Newton And His Friend Taylor

Recall that Newton's Method is stated as a recursive algorithm as follows

$$x_{n+1} = x_n - \frac{u(x_n)}{u'(x_n)}.$$

This, however, poses some issues: how do we actually perform this recursion without knowing u'? To do this, we will calculate $u(\tilde{x}_1)$ and $u'(\tilde{x}_1)$ by writing a Taylor expansion around $\tilde{x}_0$, our *starting point*.

We will need to determine a fixed number of terms of a Taylor series of u around $\tilde{x}_0$ to allow us to improve our approximation of $\tilde{x}_1$ via Newton's Method. To do this, however, we will need to know how to calculate the higher-order derivatives of u and u', which seemingly introduces additional barriers. However, to remedy this, we will assume that the p, q, s terms in (1.1) are at most second-degree polynomials (and recall that $q = p'$ by the self-adjoint property). This does not affect our overall purpose, since the Legendre differential equation given by (1.2) actually satisfies this condition. We can

then differentiate (1.1) k times to produce the following five-term recurrence relation

$$\begin{aligned} pu^{(k+2)} &= -(k+1)p'u^{(k+1)} - \left(\frac{k(k+1)}{2}p'' + s\right)u^{(k)} \\ &\quad -ks'u^{(k-1)} - \frac{k(k-1)}{2}s''u^{(k-2)}. \end{aligned} \tag{1.9}$$

Fortunately, when we restrict our attention to Legendre polynomials, this will reduce to a three-term recurrence relation, since s is constant. That said, the key observation to make here is that we can now calculate a Taylor expansion up to a fixed number of terms, thereby approximating $u(\tilde{x}_1)$ and $u'(\tilde{x}_1)$. We can then improve the approximation of $\tilde{x}_1$ by repeatedly applying Newton's Method; we need only do this at most four times (for 16-bit double precision), since Newton's Method converges quadratically, effectively doubling our accuracy at each iteration. However, there's a problem: in order to determine the Taylor expansion about $\tilde{x}_0$ by the recurrence in (1.9), we need to know the values of u''', u'', u', and u at $\tilde{x}_0$ in order to obtain $u^{(4)}(\tilde{x}_0)$ and beyond. We will address this problem when we restrict our discussion to Legendre polynomials.

4. Legendre Polynomials

At this point, we've shown that we can start with a root of u and find the next successive root in constant time! This follows from the fact that we fix the number of terms in our Taylor expansion of u and u' about $\tilde{x}_0$, while likewise fixing the number of Newton's Method iterations. This tells us that our algorithm is linear with respect to the number of roots of our solution to (1.1). However, as with any kind of induction, we need a base case, and so we need some method for identifying at least one root of u; this will, of course, depend on the individual problem. And so we will now depart from generality and hereinafter focus on Legendre polynomials.

4.1 Observations

The first observation we make is that Legendre polynomials are either even or odd. This tells us that if n is even, then $x = 0$ is a critical point, hence $P_n'(0) = 0$; likewise, if n is odd, then $P_n(0) = 0$. In both cases, this means that if $\tilde{x}$ is a root of P_n, then $-\tilde{x}$ is likewise a root! And so we need only focus our computations on the half-interval $[0, 1)$ instead of the entire interval $(-1, 1)$.

For an odd Legendre polynomial, we can simply apply the methodology described in Section 3 to "hop" from root to root (we still need to determine $P_n'(0)$, however).

For an even Legendre polynomial, we will exploit the fact that $x = 0$ is a critical point. Recall that our phase variable θ has the property that critical points occur precisely when $\theta = 0$. And so to find an approximate of our first root of P_{even}, we need only solve (1.8) with Runge-Kutta on the interval $-\pi/2 < x < 0$ with initial condition $x(0) = 0$. We then refine this approximation with Newton's Method as previously described (we still need to determine $P_n(0)$, however).

4.2 Orthogonality

Recall that Newton's Method required us to calculate a Taylor series expansion around our initial root $\tilde{x}_0$. This expansion exploits a three-term recurrence relationship given by (1.9) with s constant, and so to calculate $u''(\tilde{x}_0)$ and beyond, we need to independently determine $u(\tilde{x}_0)$ and $u'(\tilde{x}_0)$. Of course, since $\tilde{x}_0$ is assumed to be a root of u, then $u(\tilde{x}_0) = 0$; however, we still need to determine the value of $u'(\tilde{x}_0)$. To do this, we will need to exploit orthogonality of the Legendre polynomials on the interval $[-1, 1]$.

In general, given a family of orthogonal polynomials $\{q_n\}_{n=0}^{\infty}$ over an interval $[a, b]$, we can express the higher order polynomials via the recurrence

$$a_n q_{n+1}(x) = (b_n + c_n x)q_n(x) - d_n q_{n-1}(x), \qquad \text{for all } n \geq 1,$$

for some coefficients a_n, b_n, c_n, d_n. For Legendre polynomials, we have

$$a_n = n + 1, \qquad b_n = 0, \qquad c_n = 2n + 1, \quad \text{and} \qquad d_n = n,$$

which translates to the recurrence relation

$$(n + 1)P_{n+1}(x) = (2n + 1)xP_n(x) - nP_{n-1}(x). \tag{1.10}$$

where $P_0(x) = 1$ and $P_1(x) = x$. Differentiating this recurrence allows us a way to define a recurrence

$$(n + 1)P'_{n+1}(x) = (2n + 1)xP'_n(x) - nP'_{n-1}(x) + (2n + 1)P_n(x). \tag{1.11}$$

between their derivatives.

4.3 Simplification

The importance of the recurrence relations in (1.10) and (1.11) are that they allow us to start the process of finding the first nontrivial root of both even and odd Legendre polynomials. In the case of $P_{n=\text{even}}$, we calculate $P_n(0)$ via (1.10); and in the case of $P_{n=\text{odd}}$, we calculate $P'_n(0)$ via both recurrence relations. Moreover, we need only do this for the first iterate of our algorithm! Reason being that we calculate $u'(\tilde{x}_1)$ via Taylor expansion about $\tilde{x}_0$, and so we can simply store this value in memory for when we expand around $\tilde{x}_1$ to better approximate $\tilde{x}_2$, and so forth. This allows us to restrict our concern with (1.10) and (1.11) to the case when $x = 0$, i.e.

$$P_{n+1}(0) = \frac{n}{n+1} P_{n-1}(0)$$

and

$$P'_{n+1}(0) = -\frac{n}{n+1} P'_{n-1}(0) + \frac{2n+1}{n+1} P_n(0).$$

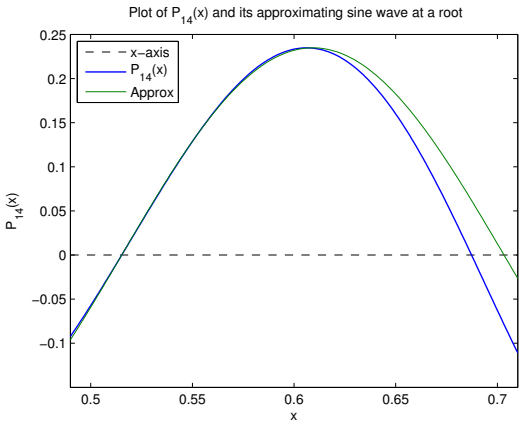
4.4 Another Simplification

The purpose of applying a Prüfer Transformation to our original ODE and then solving (1.7) with Runge-Kutta was to approximate roots by "hopping" from one root to the next. While solving this IVP requires mild computation, we can actually replace it by another, more simple method. The idea behind this simplification is to observe that Legendre polynomials look locally like a sine wave! And so by approximating the local wavenumber k near our root $\tilde{x}_0$, we can obtain a rough estimate for the subsequent root by

$$\tilde{x}_1 = \tilde{x}_0 + \pi/k.$$

We suppose that $P_n(x) \approx a\sin(kx)$ in a neighborhood of $\tilde{x}_0$ for some amplitude a. Differentiating, we find that $P_n'(x) \approx ak\cos(kx)$ and $P_n'''(x) \approx -ak^3\cos(kx)$, and so $P_n'(\tilde{x}_0) = ak$ and $P_n'''(\tilde{x}_0) = -ak^3$. We can then determine the approximate wavenumber by

$$k = \sqrt{-P_n'''(\tilde{x}_0)/P_n'(\tilde{x}_0)}.$$



In fact, this is far easier to compute via the recurrence in (1.9) than by solving (1.8) to obtain our initial estimate to our next root. After all, our only concern at this stage is in finding a rough estimate to our next root, to which will refine by applying Newton's Method to a Taylor expansion around our existing root.

5. Gaussian Quadrature

After all this talk of finding the roots of $P_n(x)$ to determine Gaussian quadrature nodes, you may be asking yourself, "what about the quadrature weights?" Well, these are determined by a simple formula

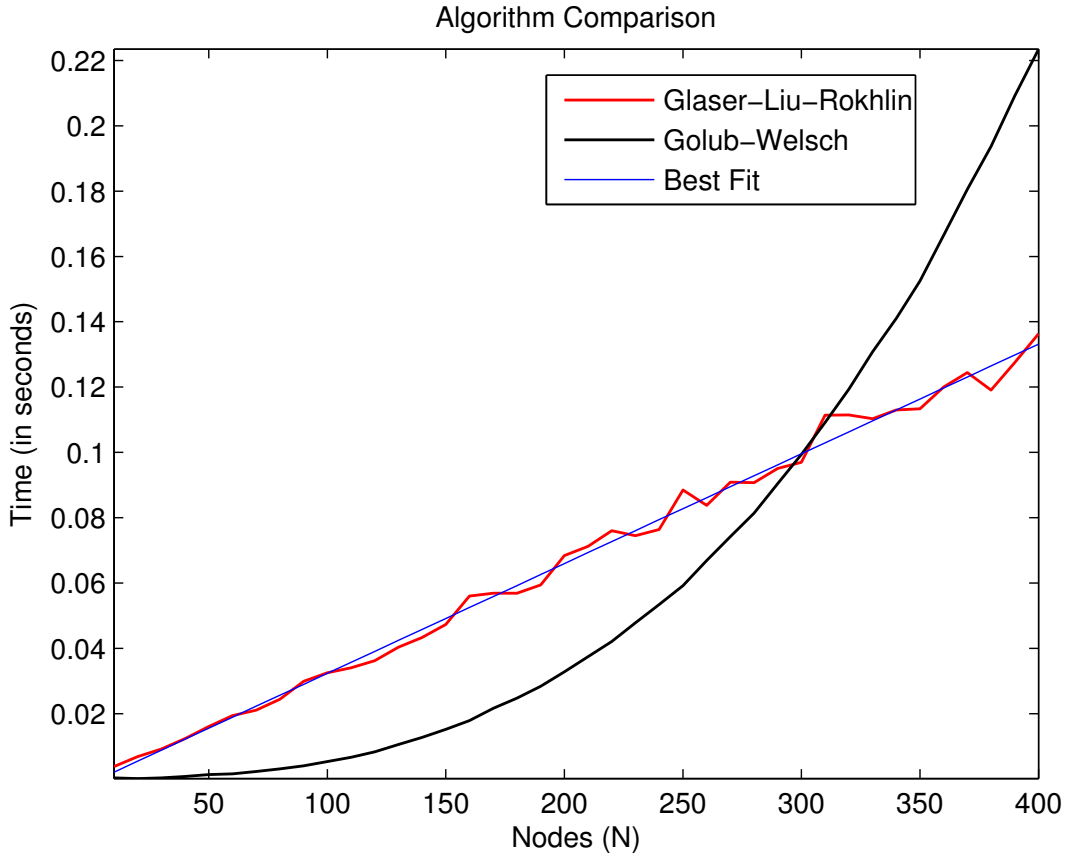
$$w_k = \frac{2}{1-x_k^2} \cdot \left(\frac{1}{P_n'(x_k)}\right)^2, \tag{1.12}$$

which are easy to calculate and store as we discover each successive root of P_n. This equation can be derived from some general theory regarding quadrature and more specific properties of Legendre polynomials. The general outline for deriving this formula can be found in [3].

6. Comparison

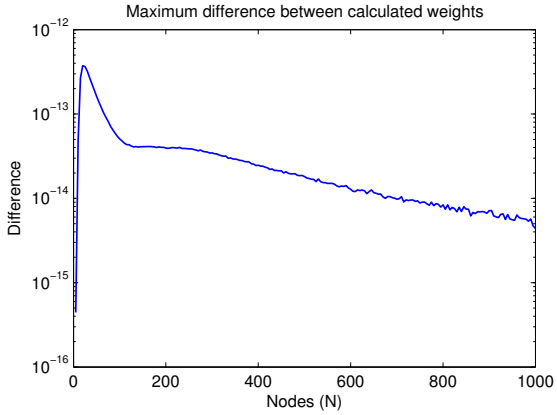
In this section, we compare the Glaser-Liu-Rokhlin algorithm to the Golub-Welsch algorithm. As a linear algorithm, we expect GLR to outperform GW, which is certainly

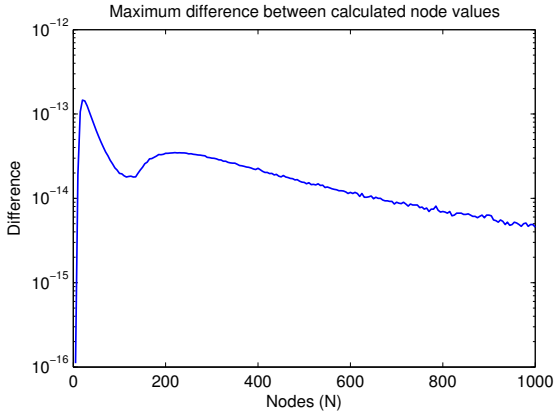
the case as the figure below demonstrates.



For less than three-hundred nodes, it is still preferable to use the classical GW method; however, for a larger number of nodes, GLR clearly outperforms GW. In fact, a naïve implementation of GW quickly becomes intractable at around 1000 nodes, where as GLR can very easily calculate 25000 nodes.

Lastly, we compare the numerical value of the nodes and weights found by these two methods.





The comparison here is a direct comparison between the nodes and weights found by both the GLR and GW method, and the maximal such difference was recorded. This tells us that the difference between the calculated nodes and weights are very close to being on the order of machine precision. In effect, GLR method is a viable replacement for the classical GW method for calculating a large number of quadrature nodes.

References

[1] G. Birkhoff, G.-C. Rota, *Ordinary Differential Equations*. Ginn, Boston, 1962, Chap. 10.5.

[2] A. Glaser, X. Liu and V. Rokhlin, "A fast algorithm for the calculation of the roots of special functions", SIAM Journal on Scientific Computing, 29 (2007), 1420-1438.

[3] F. B. Hildebrand, *Introduction to Numerical Analysis*. New York: McGraw-Hill, pp. 323-325, 1956.