Revised

NUMERICAL ANALYSIS OF LOVE'S INTEGRAL EQUATION

by

Amy Pilkington

Advisor: Andre Weideman

REU93, OSU

I. Introduction

This paper is the result of analyzing and comparing four separate ways of numerically solving Love's integral equation. Presented first is *Nystrom's* method of solving integral equations using three different approaches: (1) the Trapezoidal rule, (2) Simpson's rule, and (3) the Legendre rule. The final method presented is known as the *degenerate kernel* method. Each method is programmed in Matlab and the solutions found by each are compared, but more informative are their differences in the actual errors and the calculated error bounds. We will want to find which methods and approaches are most accurate and/or most efficient in approximating the true solution of Love's integral equation. Intuitively, these results can be used to predict how well these methods would numerically solve any similar type of integral equation. First, we will do a brief review of how the three rules mentioned above can also be used to numerically approximate the value of an integral.

II. Numerical Integration

There are many examples of integrals that can not be evaluated explicitly, or if they can, it is not quite efficient to do so. This is where it becomes useful to numerically approximate the integral by a process known as *numerical integration*, or *quadrature*. This method is mainly used to numerically integrate a function, $f(s)$, which maps the set of real numbers to itself, on a finite interval $[a, b]$. The integral is approximated by a linear combination of values of this function at certain nodal points, $s_1, s_2, ..., s_n$, with coefficients, or weights, $w_1, w_2, ..., w_n$, such that

$$Q(f(s)) = \int_a^b f(s)\,ds \approx w_1 f(s_1) + w_2 f(s_2) + \cdots + w_n f(s_n) = Q_n(f(s)). \qquad (1)$$

The three rules mentioned above are referred to as the numerical quadrature rules when used to approximate such an integral this way. The following is an explanation of how this process is applied according to each rule.

With the Trapezoidal rule, we let $n = N+1$, where N is an even number, and we partition the interval $[a, b]$ into N equally spaced subintervals with endpoints, $s_1, s_2, ..., s_{N+1}$, where

$s_1 = a$, $s_{N+1} = b$, and $s_{j+1} > s_j$, $j = 1, ..., N+1$. These endpoints are the nodal points where the integrand, $f(s)$, is evaluated for the approximation in equation (1). To approximate the area under $f(s)$, we sum the areas of the trapezoids, each with height, $h = s_{j+1} - s_j$, $j = 1, ..., N$, which is the same as $h = (b-a)/N$, and with bases, $f(s_1), ..., f(s_{N+1})$. Thus, remembering that the formula of the area of a trapezoid is $(\frac{1}{2}h)(base_1 + base_2)$, each base is added twice in the summing of these except for $f(s_1)$ and $f(s_{N+1})$, and the weights of equation (1) are $w_i = h/2$ for $i = 1, N+1$, and $w_i = h$ for $i = 2, ..., N$. The formula for the theoretical error, $Q(f(s)) - Q_n(f(s))$, in using the Trapezoidal rule is

$$E_n(f) = -\frac{(b-a)h^2}{12} f''(\alpha) \leq \gamma_n \|f''\|_\infty, \ \ \alpha \in [a, b],$$

as shown in Atkinson [1989], and where

$$\gamma_n = \frac{(b-a)h^2}{12}.$$

Notice that since the second derivative of any constant or linear $f(s)$ is zero, this error also equals zero for any such function. Therefore, the Trapezoidal rule finds the exact integral for these two kinds of functions.

With Simpson's rule, as in the Trapezoidal rule, we do the same partition of the interval, $[a, b]$, into an even number, N, of equally spaced subintervals with endpoints, $s_1, ..., s_{N+1}$, that are again the same nodal points as before. We then approximate $f(s)$ on each interval, $[s_j, s_{j+2}]$, $j = 1, \ldots, N-1$, with a quadratic interpolating polynomial and sum their respective integrals. The corresponding weights are then $w_i = h/3$ for $i = 1, N+1$, $w_i = 4h/3$ for $i = 2, 4, ..., N$, and $w_i = 2h/3$ for $i = 3, 5, ..., N-1$, where $h = s_{j+1} - s_j$, $j = 1, \ldots, N$, or $h = (b-a)/N$, as in the Trapezoidal rule. The formula for the theoretical error in using Simpson's rule is

$$E_n(f) = -\frac{(b-a)h^4}{180} f^{(4)}(\alpha) \leq \gamma_n \|f^{(4)}\|_\infty, \ \ \alpha \in [a, b],$$

also in Atkinson [1989], and where

$$\gamma_n = \frac{(b-a)h^4}{180}.$$

Notice that this error equals zero for any function that is of degree ≤ 3, and therefore Simpson's rule finds the exact integral of any such function.

In both of the above rules, the nodal points are equally spaced apart. It might be more accurate to approximate the integral with a linear combination of values of the integrand at nodal points that aren't equally spaced apart. The Legendre rule uses this technique. To illustrate, we will show how it works for $n = 2$, $a = -1$, and $b = 1$. For any other finite interval, one makes a linear transformation so that one has an integral with these two limits. Then we have

$$Q(f(s)) = \int_{-1}^{1} f(s)\, ds \approx w_1 f(s_1) + w_2 f(s_2),$$

and we have four unknowns to solve for. We need to set up a system of four of these equations. We choose the following four functions and use their corresponding equations to create the system:

$$\begin{array}{ll} f_1(s) = c(constant) & Q_1 = 2c = w_1 c + w_2 c, \\ f_2(s) = s & Q_2 = 0 = w_1 s_1 + w_2 s_2, \\ f_3(s) = s^2 & Q_3 = \frac{2}{3} = w_1 s_1^2 + w_2 s_2^2, \\ f_4(s) = s^3 & Q_4 = 0 = w_1 s_1^3 + w_2 s_2^3, \end{array}$$

with solution $w_1 = w_2 = 1$, and $s_1 = -\sqrt{\frac{1}{3}}$, $s_2 = \sqrt{\frac{1}{3}}$. We chose these functions so that we could have exact solutions for up to as high a degree as possible. This process is done similarly for any n with a system of $2n$ equations. In fact, the nodal points are the zeros of the n^{th}-degree Legendre polynomial. Data tables can be found which contain these weights and nodal points already calculated for certain n. We will use a Matlab program, found in the Netlib network, to calculate these values for any n. The theoretical error formula for the Legendre rule is

$$E_n(f) = \frac{2^{2n+1}(n!)^4}{(2n+1)[(2n)!]^2} \frac{f^{(2n)}(\alpha)}{(2n)!} \leq \gamma_n \|f^{(2n)}\|_\infty, \quad \alpha \in [-1,1],$$

as in Atkinson [1989], and where

$$\gamma_n = \frac{2^{2n+1}(n!)^4}{(2n+1)[(2n)!]^3}.$$

Notice, as before, that this error equals zero for any function of degree $\leq 2n-1$. Therefore, the Legendre rule is increasingly exact as n increases.

III. Nystrom's method for solving Love's integral equation

Slightly more complicated is the solving of integral equations numerically. For example, one common type of integral equation is the *Fredholm integral equation of the second kind*,

$$((I-K)u)(s) = u(s) - \int_a^b k(s,t)u(t)\,dt = g(s), \quad a \leq s \leq b. \tag{2}$$

For our purposes, we are given $g \in C[a,b]$, the set of continuous functions on $[a,b]$ with maximum norm, and $k(s,t)$, a continuous kernel defined for $s,t \in [a,b]$, and we want to find $u \in C[a,b]$. Also, K is a continuous mapping from $C[a,b]$ into itself and I is the identity mapping. In comparing the three rules of Nystrom's method, we will use Love's integral equation,

$$u(s) - \frac{d}{\pi} \int_{-1}^{+1} \frac{u(t)}{d^2 + (s-t)^2}\,dt = 1 \equiv g(s), \quad -1 \leq s \leq 1, \quad (I)$$

which is of the aforementioned form and is encountered in electromagnetics where -d equals the distance between two charged disks. Since the exact solution of this Love's integral

equation with $g(s) = 1$ is not known, it will be helpful in comparing these rules to also solve Love's integral equation with $g(s) = \arctan(1+s) + \arctan(1-s)$,

$$u(s) + \frac{1}{\pi}\int_{-1}^{+1} \frac{u(t)}{d^2 + (s-t)^2}\, dt = \arctan(1+s) + \arctan(1-s) \equiv g(s), \quad -1 \le s \le 1. \qquad (II)$$

This equation was found by setting $u(s) = 1$ for all $s \in [-1, 1]$, $d = -1$, and solving exactly for $g(s)$ by trigonometric substitution. Therefore, we know the exact solution beforehand, and otherwise, we suppose the same conditions as detailed for the previous equation. For this analysis, we will only consider $d = -1$ in both types of Love's integral equation.

The following is a general explanation of how Nystrom's method solves Love's integral equation. The approach is to set up a linear system of n equations where each equation has j fixed as $j = 1, ..., n$ respectively. Each equation is of the form,

$$u_j + (K_n u)(s_j) = u_j + \frac{1}{\pi}\sum_{l=1}^{n} w_l \frac{u_l}{1 + (s_j - t_l)^2} = g(s_j), \quad j = 1, \ldots, n, \qquad (3)$$

where K_n is the linear mapping from $C[a, b]$ into itself defined in this equation, $s_i = t_i$ are the nodes and w_i are the weights for the particular rule being applied, and $u_i \approx u(s_i)$. Therefore, for the terms in the summation, $u_l \approx u(t_l)$, and s_j is fixed, and so we can apply each numerical quadrature rule as if integrating a function only of t with respect to t. With $\phi(s_j, t_l) = \frac{1}{\pi(1+(s_j-t_l)^2}$, the matrix form of the system is

$$\begin{bmatrix} 1 + w_1\phi(s_1,t_1) & w_2\phi(s_1,t_2) & \ldots & w_n\phi(s_1,t_n) \\ w_1\phi(s_2,t_1) & 1 + w_2\phi(s_2,t_2) & \ldots & w_n\phi(s_2,t_n) \\ \vdots & & & \\ w_1\phi(s_n,t_1) & w_2\phi(s_n,t_2) & \ldots & 1 + w_n\phi(s_n,t_n) \end{bmatrix} \begin{bmatrix} u_1 \\ u_2 \\ \vdots \\ u_n \end{bmatrix} = \begin{bmatrix} g(s_1) \\ g(s_2) \\ \vdots \\ g(s_n) \end{bmatrix}.$$

Using this system for each rule applied to each of the Love's integral equations, (I) and (II), Matlab was programmed to solve for the vector $(u_1, ..., u_n)$. Each rule was applied for $N(n$ for Legendre$) = 4$, 8, 16, 32, and 64. Figure 1 shows the plots of the resulting solutions as u_i vs. s_i, $i = 1, ..., n$, for equation (II) where $u(s)$=1. Notice the chosen n for each rule shows at what n value the approximated solution goes to the exact solution, that is as far as Matlab is concerned since Matlab rounds off at 16 significant figures. This shows, that for solving Love's integral equation, the Legendre rule is the superior method and that Simspon's is better than Trapezoidal since they require a smaller n to reach this solution. This is why it is helpful to also be solving for equation (II) where we know the exact solution and can assume the same level of accuracy by each rule in solving for equation (I), even though we don't know its exact solution. Another way to see which rule is more accurate is in Figure 2. Here we have plotted $(u_i - 1)$ vs. s_i for certain n. Notice that the error gets smaller for equal or less n as you go from Trapezoidal to Simpson's to Legendre. We only went to $n = 16$ for the Legendre rule because at the next n the error becomes too small for Matlab accuracy.

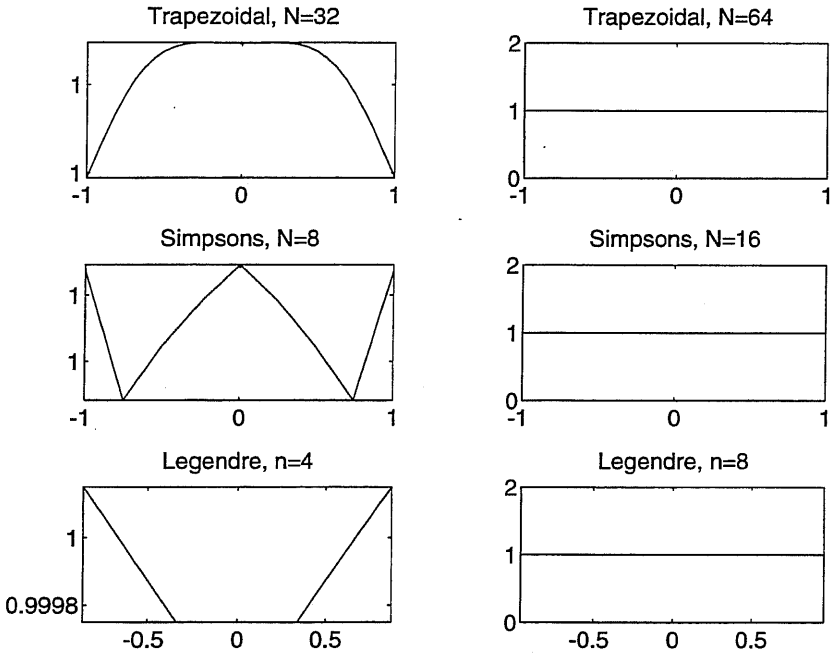

Figure 1 Solutions for equation (II), u_j vs. s_i, $i = 1, ..., n$.

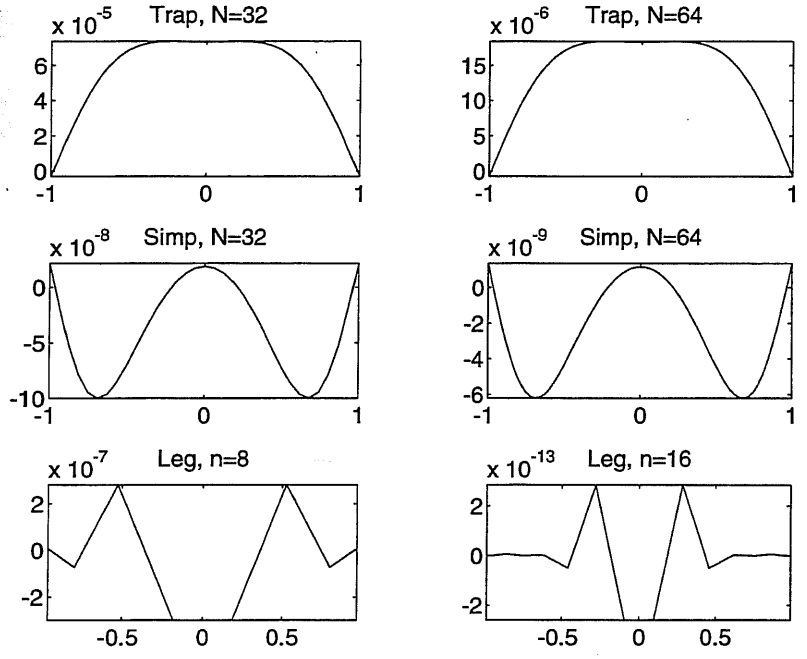

Figure 2 Actual error for equation (II), $(u_i - 1)$ vs. s_i, $i = 1, ..., n$.

Another way to compare the actual errors for each rule is to calculate the euclidean norm of the difference between the solution vector, $(u_1, ..., u_n)$ and the exact solution vector, $(u(s_1), ..., u(s_n))$. For equation (II), the exact solution vector is (1,1,1,...,1). For equation (I), we use the solution vector found using Simpson's rule with n=256 as the exact solution vector. This is close enough to the true solution for our purposes. In Figure 3, the solid lines are the plots of the log of the actual error vs. n for each rule solving equation (II), and similarly in Figure 4 for equation (I). There is no plot of the actual error for the Legendre rule in solving equation (I). This is because between using different n values there are no nodal points in common, and so we can not take the difference between their solution vectors to obtain an error vector. This author believes there should be a way around this problem and will continue to research the possibilities. Notice the difference of the rate of convergence between each rule. Again, the Legendre rule shows superiority since it converges to zero the fastest, and Simpson's rule converges faster than the Trapezoidal. Also note that the values less than 10^{-13} in Figure 3 are not trustworthy because this is where Matlab loses its accuracy. The line would really continue down at the same slope it has before this point. The Matlab program that solves for equation (II) using the Legendre rule is in the appendix of this paper.

Since we know the exact solution of equation (II) to be $u(s) = 1$, we can add a special feature to the error analysis of each rule in solving for this equation. We can replace $u(s)$ and $u(t)$ with 1, and set $(s - t) = w$ so that the equation becomes

$$1 + \int_{s+1}^{s-1} f(w)\, dw = \arctan(1 + s) + \arctan(1 - s),$$

where

$$f(w) = \frac{1}{\pi}\frac{1}{1 + w^2} = \frac{-1}{2\pi i}\left(\frac{1}{w + i} - \frac{1}{w - i}\right).$$

By differentiating this right side p times, we find that

$$f^{(p)}(w) = -\frac{p!}{2\pi i}\left[\frac{1}{(w + i)^{p+1}} - \frac{1}{(w - i)^{p+1}}\right].$$

Then let $w + i = re^{i\theta}$ and $w - i = re^{-i\theta}$, so we have

$$(w + i)^{-(p+1)} = r^{-(p+1)}e^{-i(p+1)\theta}, \quad (w - i)^{-(p+1)} = r^{-(p+1)}e^{i(p+1)\theta}, \quad \textit{and}$$

$$f^{(p)}(w) = -\frac{p!}{2\pi i r^{(p+1)}}[-2i \sin(p + 1)\theta] = \frac{p!}{\pi(\sqrt{w^2 + 1})^{(p+1)}} \sin\left[(p + 1) \arctan\frac{1}{w}\right].$$

Therefore,

$$\max_{-\infty \le w \le \infty} | f^{(p)}(w) | \le \frac{p!}{\pi} = M_p. \tag{4}$$

Since here we know what the integrand function is explicitly, we can then use the error formulas from numerical integration stated already for the three numerical quadrature rules to find the corresponding cheating error bounds in solving the original integral equation (II) by replacing $\|f^{(q)}\|_\infty$ with $\frac{q!}{\pi}$ and setting $(b - a) = 2$. They are cheating error bounds because

you are not supposed to already know the solution of the equation when computing the true theoretical error bounds. These values will simply be used for their informative value in our comparative analysis. In Figure 3, the log of these cheating error bounds is plotted vs. n as a dash-dot-dash line. As expected, the cheating error bound is just a bit greater than the actual error, and the rate of convergence is the same as the rate for the actual error.

An honest theoretical error bound for Nystrom's method is

$$\|u - u_n\| \leq \frac{\|(I-K)^{-1}\|\|(K_n - K)g\| + \delta\|(I-K)^{-1}g\|}{1-\delta}, \tag{5}$$

where

$$\delta \equiv \|(I-K)^{-1}(K_n - K)K_n\|,$$

as in Cryer [1982]. Here u_n is the solution vector found according to the rule being applied and K and K_n are as defined in equations (2) and (3) respectively. As found in Cryer [1982],

$$\|K\|_\infty = \max_{-1\leq s\leq 1} \int_{-1}^{1} |k(s,t)|\, dt = \frac{1}{2},$$

and therefore

$$\|(I-K)^{-1}\| \leq \frac{1}{1-\|K\|_\infty} \leq 2.$$

In Cryer [1982] he shows how to find the rest of the bounds of the terms in the error formula (5) for $n = 3$. The following is a generalization of the same process for any n. From the error bounds for numerical integration mentioned above and from the bound in (4) we find

$$\|(K_n - K)g\| = \max_{-1\leq s\leq 1} |Q_n(k(s,t) - Q(k(s,t))|\|g\|_\infty,$$

$$\leq \gamma_n M_q \|g\|_\infty.$$

Similarly,

$$\|(K_n - K)K_n\|_\infty \leq (q+1)\frac{\gamma_n}{\pi} M_q.$$

For $g = 1$, $\|g\|_\infty = 1$, and for $g(s) = \arctan(1+s) + \arctan(1-s)$, $\|g\|_\infty = 1.5$. By placing all of these bounds into the above error bound formula(5), we can obtain the log of the theoretical error bounds for each n plotted in Figures 3 and 4 by the dashed line. Notice that for each rule the rate of convergence is precisely in line with that of the actual error as expected. It is also worthwhile to note how close this theoretical error bound actually is to the true error, showing its credibility as a useful bound.

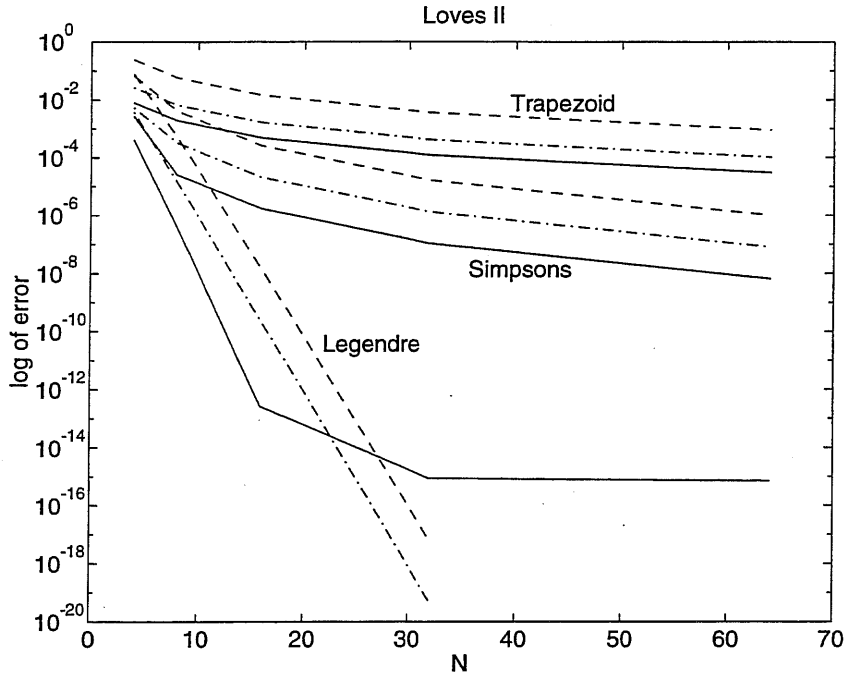

Figure 3 Error analysis for equation (II).

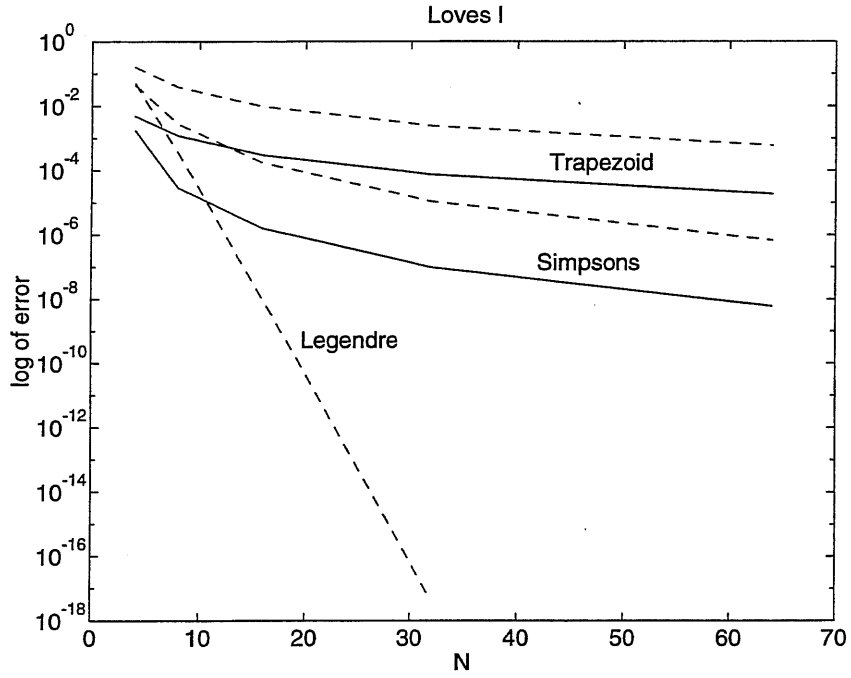

Figure 4 Error analysis for equation (I).

IV. Degenerate Kernel method

In the degenerate kernel method we approximate equation (2) by

$$((I - K_n)u_n)(s) = u_n(s) - \int_a^b k_n(s,t)u_n(t)\,dt = g(s), \tag{6}$$

where $k_n(s,t)$ is the appropriately chosen degenerate kernel,

$$k_n(s,t) = \sum_{i=1}^{n} x_i(s)y_i(t). \tag{7}$$

The functions $x_1, ..., x_n$ must be a basis of the range of K_n, which is the n-dimensional subspace of $C[a,b]$. The equations used in solving for $u_n(s)$ are the following:

$$\begin{array}{ll} (a) & u_n(s) = g(s) + \sum_{i=1}^{n} w_i x_i(s), a \le s \le b, \\ (b) & Aw = c, \\ (c) & c = (c_i) = (\int_a^b y_i(t)g(t)\,dt), \\ (d) & A = (a_{ij}) = (\delta_{ij} - \int_a^b y_i(t)x_j(t)\,dt), \end{array}$$

where

$$\delta_{ij} = \begin{cases} 1 & \text{if } i = j \\ 0 & \text{if } i \neq j \end{cases}$$

In Cryer [1982], he shows the process of using the degenerate kernel method to solve for u_3. The following is a generalization of the same process for any odd number n. The first job is to find the best mth degree polynomial approximation to $\frac{1}{1+v}$ on $C[0,4]$,

$$p_m(v) = a_N v^m + a_{N-1}v^{m-1} + \cdots + a_2 v + a_1,$$

where $m = \frac{n-1}{2}$ and $N = m+1$. This can be done in Matlab using a function called *minimax*. Since $s, t \in [-1,1]$, then $(s-t)^2 \in [0,4]$. Therefore, with

$$k(s,t) = \frac{-1}{\pi}\frac{1}{1+(s-t)^2}$$

as in equations (I) and (II), we have

$$k(s,t) \approx k_n(s,t) = \frac{-1}{\pi}p_m((s-t)^2),$$

which expanded becomes equation (7) with

$$
\begin{aligned}
&\pi x_1(s) = 1, && y_1(t) = -a_N t^{2m} - a_1/2,\\
&\pi x_2(s) = a_N \binom{2m}{1} s, && y_2(t) = t^{2m-1},\\
&\pi x_3(s) = -a_N \binom{2m}{2} s^2 - a_{N-1}, && y_3(t) = t^{2m-2},\\
&\pi x_4(s) = a_N \binom{2m}{3} s^3 + a_{N-1} \binom{2m-2}{1} s, && y_4(t) = t^{2m-3},\\
&\pi x_5(s) = -a_N \binom{2m}{4} s^4 - a_{N-1} \binom{2m-2}{2} s^2 - a_{N-2}, && y_5(t) = t^{2m-4},\\
&\pi x_6(s) = a_N \binom{2m}{4} s^5 + a_{N-1} \binom{2m-2}{3} s^3 + a_{N-2} \binom{2m-4}{1} s, && y_6(t) = t^{2m-5},\\
&\vdots && \vdots\\
&\pi x_n(s) = -a_N \binom{2m}{2m} s^{2m} - a_{N-1} \binom{2m-2}{2m-2} s^{2m-2} - \\
&\quad a_{N-2} \binom{2m-4}{2m-4} s^{2m-4} - \cdots - a_1 \binom{2m-2m}{2m-2m} + a_1/2, && y_n(t) = 1.
\end{aligned}
$$

The theoretical error bound using the degenerate kernel method to approximately solve Love's integral equation is

$$\|u - u_n\| \leq \|(I - K)^{-1}(K - K_n)u_n\|, \tag{8}$$

where K_n is now as defined in equation (6). As before,

$$\frac{1}{1 - \|K\|_\infty} \leq 2,$$

Also,

$$\|K - K_n\|_\infty = \max_{-1 \leq s \leq 1} \int_{-1}^{1} |k(s,t) - k_n(s,t)| \leq \frac{2\beta}{\pi},$$

where $\beta = \max_{0 \leq v \leq 4} \|p_m(v) - \frac{1}{1+v}\|$. To find β, we set $e_m(v) = p_m(v) - \frac{1}{1+v}$ and set its derivative equal to zero to find

$$
\begin{aligned}
e'_m(v) \equiv 0 \;=\; & m a_N v^{m+1} + (2m a_N + (m-1)a_{N-1})v^m \\
& + (m a_N + 2(m-1)a_{N-1} + (m-2)a_{N-2})v^{m-1} \\
& + ((m-1)a_{N-1} + 2(m-2)a_{N-2} + (m-3)a_{N-3})v^{m-2}, \\
& + \ldots, + (3a_4 + 2 \cdot 2a_3 + a_2)v^2 \\
& + (2a_3 + 2a_2)v + a_2 + 1.
\end{aligned}
$$

We then compare the roots of this equation with $e(0)$ and $e(4)$ in order to find β as the maximum of their absolute values. The only bound remaining to be evaluated is $\|u_n\|_\infty$, but after using this method to solve for equations (I) and (II), we see that it is less than or

equal to $u_n(1)$ in both cases. Therefore we have all of the required bounds that we need to substitute into formula(8) so to arrive at a theoretical error bound that depends on m.

The Matlab programs that follow this degenerate kernel method of solving Love's integral equation can be found in the appendix of this paper. In these programs there are some functions that need to be explained since they are not already defined Matlab functions. The *intp* and *intpg* functions evaluate the integral of the inputed function. The *bic* function is the coefficient, *n choose k*, and the *gtan* function evaluates $(ua + g)$ at s. These programs are not complete because they do not compute the actual error or error bound. This is future implementation that the author will be working on. Figure 5 shows the plot of the solution for both equations at $m = 1$ and $m = 4$. If you compare the plot for equation (II), $m = 4$, with the plot of the Legendre rule for the same equation, $n = 4$, in Figure 1, you'll notice that their respective errors from the true solution, $u = 1$, is about 2×10^{-4} for both. Of course for a better comparison, we will want to plot the degenerate kernel method's actual error after these values are computed. The important feature of these plots is that their solutions closely agree with what we found using Nystrom's method.

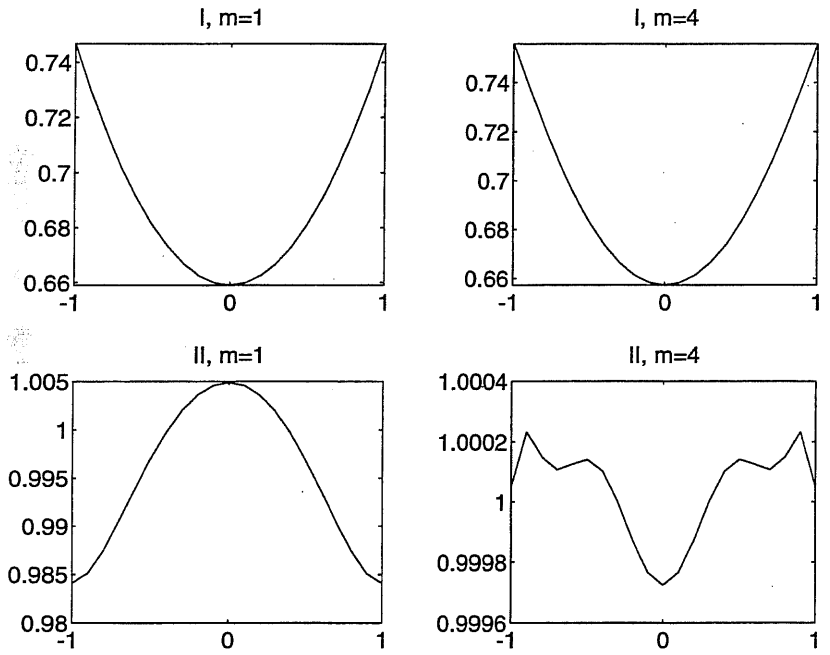

Figure 5 Solutions of degenerate kernel method.

V. Conclusion

In conclusion, we find that the Legendre rule is definitely the preferred rule of Nystrom's method in solving Love's integral equation. The degenerate kernel method seems to be of comparable accuracy, which will be more clear when the error analysis of this method is complete. Even so, the degenerate kernel method is not as efficient due to the number of steps it takes to find the approximate polynomial. There are other methods for solving these

integral equations that could be compared to these, but it would be surprising if any were more accurate and efficient than the Legendre rule.

VI. References

1. Atkinson, Kendall E., *An Introduction to Numerical Analysis*, John Wiley and Sons, New York, 1989.

2. Cryer, Colin W., *Numerical Functional Analysis*, Clarendon Press, Oxford, 1982.

APPENDIX

```
%
%
%
%  Matlab program to solve Equation (II) by the Legendre rule
%  Amy Pilkington, REU93

N = input('N=?')
ker = '(1/pi)*(1+(ss-tt).^2).^(-1)';

[s,w] = gauss(N); t = s;

[ss,tt] = meshgrid(s,t); ss = ss'; tt = tt';

K = eval(ker);

W = diag(w);

A = eye(N)+K*W;

g = 1 + (1/pi)*(atan(1-s)+atan(1+s)); g = g';

u = A\g;

%  ERb is the theoretical error bound
M = prod(1:2*N);
Eb = 2.^(2*N+1)*(prod(1:N).^4)/((2*N+1)*(prod(1:2*N).^3));
b1 = (3/2/pi)*Eb*M; b2 = (2*(2*N+1)*Eb*M)/(pi.^2);
ERb = (2*b1 + 3*b2)/(1 - b2);

%  ceb is the cheating error bound
ceb = Eb*prod(1:2*N)/pi;

plot (s,u);
```

```
%  Matlab program to solve equation (I) by degenerate kernel method
%  Amy Pilkington, REU93

m = input('m=?')
n = m+1; N = 2*m+1;
a0=[1];
for q=1:m
        a0=[0 a0];
        a0=minimax('degen',a0);
end
a = minimax('degen',a0);
a = fliplr(a);
I = eye(N);
y1 = -a(n)*I(1,:)-(a(1)/2)*I(N,:);
c(1) = intp(64,y1);
for k = 2:N
        c(k) = intp(64,I(k,:));
end
B(1,:) = (1/pi)*I(N,:);
for i = 2:N
        A(i,1) = -intp(64,conv(I(i,:),B(1,:)));
end
A(1,1) = -intp(64,conv(y1,B(1,:)));
for j = 2:N
        x = (-1).^j*a(n)*bic(2*m,j-1)*I((N-j+1),:);
        l = 1;
        while j-2*l>0
                x = x+(-1).^j*a(n-l)*bic(2*m-2*l,j-(2*l+1))*I((N-j+2*l+1),:);
                l = l + 1;
        end
        if j==N
                x = x + (a(1)/2)*I(N,:);
        end
        A(1,j) = -intp(64,conv(y1,((1/pi)*x)));
        B(j,:)=(1/pi)*x;
end
for i=2:N
        for j=2:N
                A(i,j) = -intp(64,conv(I(i,:),B(j,:)));
        end
end
for i=1:N
        for j=1:N
                if i==j
                        A(i,j) = 1 + A(i,j);
                end
        end
end
w=A\c';
u = I(N,:) + w(1)*B(1,:);
for p = 2:N
        u = u + w(p)*B(p,:);
end


S = (-1:.1:1); U = polyval(u,S)
plot(S,U)
```

```
%  Matlab program to solve equation (II) by degenerate kernel method
%  Amy Pilkington, REU03

m = input('m=?')
n = m+1; N = 2*m+1;
a0 = [1];
for q=1:m
        a0=[0 a0];
        a0=minimax('degen',a0);
end
a = minimax('degen',a0);
a = fliplr(a);
I = eye(N);
y1 = -a(n)*I(1,:)-(a(1)/2)*I(N,:);
c(1) = intpg(64,y1);
for k = 2:N
        c(k) = intpg(64,I(k,:));
end
B(1,:) = (1/pi)*I(N,:);
for i = 2:N
        A(i,1) = -intp(64,conv(I(i,:),B(1,:)));
end
A(1,1) = -intp(64,conv(y1,B(1,:)));
for j = 2:N
        x = (-1).^j*a(n)*bic(2*m,j-1)*I((N-j+1),:);
        l = 1;
        while j-2*l>0
                x = x+(-1).^j*a(n-l)*bic(2*m-2*l,j-(2*l+1))*I((N-j+2*l+1),:);
                l = l + 1;
        end
        if j==N
                x = x + (a(1)/2)*I(N,:);
        end
        A(1,j) = -intp(64,conv(y1,((1/pi)*x)));
        B(j,:)=(1/pi)*x;
end
for i=2:N
        for j=2:N
                A(i,j) = -intp(64,conv(I(i,:),B(j,:)));
        end
end
for i=1:N
        for j=1:N
                if i==j
                        A(i,j) = 1 + A(i,j);
                end
        end
end
w=A\c';
ua = w(1)*B(1,:);
for p = 2:N
        ua = ua + w(p)*B(p,:);
end

S = (-1:.1:1); U = gtan(ua,S);
plot(S,U)
```