Rochester Institute of Technology
RIT Digital Institutional Repository

Theses

5-1-2009

Optimal interpolation grids for accurate numerical solutions of singular ordinary differential equations

Michael Margitus

Follow this and additional works at: https://repository.rit.edu/theses

Recommended Citation
Margitus, Michael, "Optimal interpolation grids for accurate numerical solutions of singular ordinary differential equations" (2009). Thesis. Rochester Institute of Technology. Accessed from

This Thesis is brought to you for free and open access by the RIT Libraries. For more information, please contact repository@rit.edu.

Optimal Interpolation Grids for Accurate Numerical Solutions of Singular Ordinary Differential Equations

Michael Margitus

Submitted in Partial Fulfillment of the
Requirements for the Degree of
Master of Science in
Computational and Applied Mathematics

School of Mathematical Sciences
College of Science
Rochester Institute of Technology
Rochester, New York

May 2009

Thesis Committee

Advisor:	David S. Ross
Committee Member 1:	George M. Thurston
Committee Member 2:	Chris Wahle
Graduate Program Director:	Hossein Shahmohamad

Abstract

Researchers are interested in three different types of error: Experimental error, truncation error, and Interpolation error. This thesis will study the last. Given a differential equation $g''(x) = f(x)$ and a fixed number of interpolation grid points, an optimization problem is formulated to minimize the difference between $f(x)$ and its interpolating function, thus reducing the error between the actual solution and the approximated solution of the ODE. Using the Nelder-Mead Simplex Method, the optimal distribution of grid points that will minimize the error between the solution g and its approximated solution will be found.

This technique will then be applied to the one dimensional light scattering equation $g_{xx} = \frac{\varepsilon_x^2}{R}$. Using the Nelder-Mead Method, the optimal interpolation grid for a given number of grid points will be found. These numerical computations will ultimately be used to give guidance to experimenters on where to take measurements for the Rayleigh Ratio R.

Contents

1 Introduction

1.1 Background

Though the applications of this problem are vast and can extend to a variety of circumstances, the origin of the problem at hand stems from the field of Biological Physics, specifically from the study of protein mixtures in the human eye.

When trying to characterize the eye thermodynamically, one of the properties that physicists study is the Gibbs free energy of the protein mixture. Traditionally, measuring the Gibbs free energy was thought of in a theoretical sense rather than in an experimental sense. Circa 1950, physicist John G. Kirkwood was studying light scattering properties in multi-component systems, specifically liquid mixtures [4]. Kirkwood derived the equivalent of the differential equation

$$\frac{g_{xx}}{\varepsilon_x^2} = \frac{1}{R}$$

as a way to predict the amount of light scattered by a mixture from a known free energy G and dialectric coefficient ε [3].

In our case, the question that Kirkwood set out to solve has been reformulated. That is, if the dialectric gradiant and the amount of light scattered, measured by the Raleigh Ratio R, are known, can the free energy be pre-

dicted? In this setting, Kirkwood's equation can be rewritten

$$g_{xx} = \frac{\varepsilon_x^2}{R}$$

The question now becomes a matter of accurately solving this differential equation. In an experimental setting, the accuracy of the solution depends on the accuracy of the data measurements. Aside from experimental error, and the truncation error which is incurred from solving the differential equation numerically, an accurate solution depends on the measurement of the Rayleigh Ratio, specifically how and where to measure the Rayleigh Ratio. When generalizing this for a differential equation $g_{xx}(x) = f(x)$, where $f(x)$ is any experimental data, we would like to know how to measure $f(x)$ accurately.

There are two components to consider when measuring data for $f(x)$, or in our case, the Rayleigh Ratio: Where to measure the data, and how long to spend measuring data at those points. It is obvious that taking longer, more frequent measurements will generate the most accurate data, but practicality won't allow us to measure our data an at infinite amount of locations. In a lab setting, there are limited resources and time. This translates into taking a finite number of measurements during a predetermined amount of time. Due to these limiting factors, careful consideration needs to be taken when making decisions about choosing locations to measure, and how to allocate our time so that numerical data errors can be reduced. To obtain the most accurate

solutions for our differential equation, we must answer the following questions first: How and where should these measurement points be chosen? And how long should be spent measuring data at these locations? This thesis will focus on the first question: Given a fixed finite number of measurements, where should these measurements be taken, at which values of x, in order to collect the most accurate data for the solution of the differential equation. As for how much time should should be spent at each point taking measurements, Dr. David Ross and Dr. George Thurston provide an analysis[4].

Measurement Time Analysis

Error is inversely proportional to data measurement time. Spending more time measuring data at each location will increase the accuracy of the numerical solution of the differential equation. Due to a finite amount of measurement time, a certain balance must be struck between the amount of time spent measuring data at each location, and the number locations to measure that data. Given a finite amout of time T, and an interpolation grid $(x_0, x_1, \ldots, x_N)$, where $x_0 < x_1 < \cdots < x_N$, we want to choose τ_j, the measurement time at each grid point x_j, in order minimize the error of the solution of the differential equation $g_{xx} = \frac{\varepsilon_x^2}{R}$. To assist in this analysis, the light scattering equation can be expressed in terms of the data measured, and the error made in those measurements as follows

$$g_{xx} = \sum_{j=0}^{N} f_j Q_j(x)$$

where f_j is the measured value at grid point x_j, and $Q_j(x)$ is defined as $Q_j(x) = \phi_j''(x)$ in which $\phi_j''(x)$ is the linear interpolation of the measured f_j values, defined as

$$\phi_j(x) = \begin{cases} c_j(x - x_0) & x \le x_{j-1} \\ c_j(x - x_0) + \frac{(x-x_{j-1})^3}{6(x_j - x_{j-1})} & x_{j-1} \le x \le x_j \\ \Gamma_j(x - x_N) + \frac{(x-x_{j+1})^3}{6(x_j - x_{j+1})} & x_j \le x \le x_{j+1} \\ \Gamma_j(x - x_N) & x_{j+1} \le x \end{cases}$$

and

$$c_j = \frac{x_{j+1} - x_{j-1}}{2(x_N - x_0)} \Big(\frac{x_{j+1} + x_j + x_{j-1}}{3} - x_N \Big)$$

$$\Gamma_j = \frac{x_{j+1} - x_{j-1}}{2(x_N - x_0)} \Big(\frac{x_{j+1} + x_j + x_{j-1}}{3} - x_0 \Big)$$

for $j = 1, 2, \ldots, N-1$, and

$$\phi_0(x) = \begin{cases} \frac{(x-x_1)^3}{6(x_0 - x_1)} & x \le x_1 \\ 0 & x_1 < x \end{cases}$$

$$\phi_N(x) = \begin{cases} 0 & x \le x_{N-1} \\ \frac{(x-x_{N-1})^3}{6(x_N - x_{N-1})} & x_{N-1} < x \end{cases}$$

To account for the error in data measurement, f will be written as

$$\begin{aligned} f &= \frac{\varepsilon_x^2}{R+\delta\rho} - \frac{\varepsilon_x^2}{R} \\ &= \Big(\frac{\varepsilon_x^2}{R}\Big)\frac{1}{1+\delta\frac{\rho}{R}} - \frac{\varepsilon_x^2}{R} \end{aligned}$$

where δ is a small constant and ρ is the error made in the measurement of R, the Rayleigh Ratio. Now,

$$\begin{aligned} g &= \sum_{j=1}^{N-1} f_j\phi_j(x) \\ &= \frac{\varepsilon_x^2}{R}\Big(1-\frac{\delta\rho}{R}\Big) - \frac{\varepsilon_x^2}{R} \end{aligned}$$

R is computed as the mean number of photons arriving at a detector per unit time. This is a random variable with Poisson distribution having mean zero. We will now take the square of the L^2 norm of the error of the free energy g.

$$\begin{aligned} ||g||_2^2 &= \int_0^1 g^2\,dx \\ &= \int_0^1 \Big(\sum_{j=1}^{N-1} f_j\phi_j(x)\Big)^2\,dx \\ &= \int_0^1 \delta^2\Big(\sum_{j=1}^{N-1} \rho_j\big(\frac{\varepsilon_x^2}{R^2}\big)_j\phi_j(x)\Big)^2\,dx \end{aligned}$$

Since R is Poisson distributed, ρ has mean zero, $\langle\rho_j\rho_k\rangle = 0$ and $\langle\rho_j^2\rangle = \frac{\zeta R_j}{\tau_j}$,

where the amount of time measured is inversly proportional to error. We introduce a constant ζ, that is dependent upon the measuring procedure. Also, to simplify notation the integral of $\phi_j^2(x)$ will be represented by λ_j. So

$$\begin{aligned} &= \int_0^1 \sum_{j=1}^{N-1} \Big(\frac{\varepsilon_x^2}{R^2}\Big)_j^2 \rho_j^2 \phi_j^2(x)\, dx \\ &= \sum_{j=1}^{N-1} \Big(\frac{\varepsilon_x^4}{R^4}\Big)_j \frac{\zeta R_j}{\tau_j} \lambda_j \end{aligned}$$

We will now minimize the above function subject to the constraint that $(\sum_{j=1}^{N-1} \tau_j) - T = 0$. To do this, a Lagrange multiplier σ will be introduced, and the equation becomes

$$\zeta \sum_{j=1}^{N-1} \Big(\frac{\varepsilon_x^4}{R^4}\Big)_j \Big(\frac{R_j}{\tau_j}\Big)\lambda_j + \sigma\Big[\Big(\sum_{j=1}^{N-1} \tau_j\Big) - T\Big]$$

To minimize this function we will differentiate with respect to each τ_j, and with respect to σ, and thus the equation becomes

$$-\zeta\Big(\frac{\varepsilon_x^4}{R^3}\Big)_j \Big(\frac{1}{\tau_j^2}\Big)\lambda_j + \sigma = 0$$

By solving for τ_j, we obtain

$$\tau_j = \sqrt{\Big(\frac{\varepsilon_x^4}{R^3}\Big)_j \Big(\frac{\lambda_j}{\sigma}\Big)\zeta}$$

Using the above equation we can rewrite our constraint and solve for $\sqrt{\frac{\zeta}{\sigma}}$,

$$\begin{aligned}
(\sum_{j=1}^{N-1} \tau_j) &= T \\
\sqrt{\frac{\zeta}{\sigma}} \sum_{j=1}^{N-1} \Big(\frac{\varepsilon_x^2}{R^{\frac{3}{2}}}\Big)_j \sqrt{\lambda_j} &= T \\
\sqrt{\frac{\zeta}{\sigma}} &= \frac{T}{\sum_{j=1}^{N-1} (\frac{\varepsilon_x^2}{R^{\frac{3}{2}}})_j \sqrt{\lambda_j}}
\end{aligned}$$

And thus

$$\tau_j = \frac{T}{\sum_{k=1}^{N-1} (\frac{\varepsilon_x^2}{R^{\frac{3}{2}}})_k \sqrt{\lambda_k}} \Big(\frac{\varepsilon_x^2}{R^{\frac{3}{2}}}\Big)_j \sqrt{\lambda_j}$$

This equation gives the exact amount of time to measure data at grid point x_j in order to minimize the error we make in approximating the solution in this way.

1.2 Mathematical Formulation

The goal for the formulation of this problem is to create a method that will be used to provide a recommendation on where to place interpolation points for accurate solutions for our differential equation. To do this, cases with known solutions will be analyzed, and conclusions will be drawn from them.

Given a differential equation $g''(x) = f(x)$ that measures a phenomenon, subject to the Dirichlet boundary conditions of $g(0) = 0$ and $g(1) = 0$, we want to accurately measure f, our data. In a laboratory setting, certain locations will be chosen and data will be measured at those points. In our case, we have a given function f which we interpolate to serve as our data measurements. Our function f will be interpolated at N points which is called an interpolation grid. We then compare the solutions of our differential equation $g''(x) = f(x)$ and our approximate differential equation $g^{*''}(x) = f^*(x)$, measuring the error, which is the difference between $g(x)$ and $g^*(x)$. It is this error which we are trying to minimize by selecting an optimum interpolation grid.

1.2.1 Interpolation

A linear interpolation function $f^*(x)$ is created by interpolating $f(x)$ at the points in an interpolation grid $G = (\beta_1, \beta_2, ..., \beta_n)$ as follows:

$$f^*(x_i) = \begin{cases} f(x_i) & x_i \leq \beta_1 \\ \frac{(\beta_i - x_i)}{\beta_i - \beta_{i-1}} f(\beta_{i-1}) + \frac{(x_i - \beta_{i-1})}{\beta_i - \beta_{i-1}} f(\beta_i) & x_i \leq \beta_i \text{ for } 2 \leq i \leq n \end{cases}$$

Now $g(x)$ and $g^*(x)$, the solution to the differential equation and the approximate differential equation, can be solved from $f(x)$ and $f^*(x)$, respectively.

1.2.2 Numerical Solutions of Differential Equations

In order to calculate $g^*(x)$, a finite difference scheme will be used. Our differential equation is defined over the interval $[0, 1]$, and we will use a finite difference grid of $m = 100$ points, with step size $h = \frac{1}{101}$. The standard centered difference approximation to the second derivative, $g''(x) \approx \frac{1}{h^2}[g(x + h) - 2g(x) + g(x - h)]$ will be used to approximate the solution of $g^{*''}(x) = f^*(x)$ as follows

$$\begin{aligned} g^{*''}(x_i) &= f^*(x_i) \\ \frac{1}{h^2}[g^*(x_i + h) - 2g^*(x_i) + g^*(x_i - h)] &= f^*(x_i) \\ g^*(x_i + h) - 2g^*(x_i) + g^*(x_i - h) &= h^2 f^*(x_i) \\ g^*_{i+1} - 2g^*_i + g^*_{i-1} &= h^2 f^*(x) \end{aligned}$$

where $g^*(x_i) \approx g_i^*$, the approximation of $g^*(x)$ at the i^{th} finite difference grid point, for $i = 2, ..., m$. This yields the system of equations $Ag^* = f^*$, written

$$\begin{bmatrix} -2 & 1 & 0 & \cdots & 0 \\ 1 & -2 & 1 & \cdots & 0 \\ 0 & 1 & -2 & \cdots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & \cdots & -2 \end{bmatrix} \begin{bmatrix} g_1^* \\ g_2^* \\ g_3^* \\ \vdots \\ g_m^* \end{bmatrix} = \begin{bmatrix} h^2 f_1^* \\ h^2 f_2^* \\ h^2 f_3^* \\ \vdots \\ h^2 f_m^* \end{bmatrix}$$

By solving this system of equations, $g^*(x)$, the approximation of $g(x)$, can be found.

1.2.3 Calculating Errors

The error ϵ is calculated by computing the analog of the L^2 norm over the interval [0,1],

$$\begin{aligned} \epsilon &= \sqrt{(g(x_1) - g^*(x_1))^2 + (g(x_2) - g^*(x_2))^2 + \cdots + (g(x_n) - g^*(x_n))^2 h} \\ &\approx ||g(x) - g^*(x)||_2 = \Big[\int_0^1 (g(x) - g^*(x))^2 dx \Big]^{\frac{1}{2}} \end{aligned}$$

It should be noted that using other p-norms will slightly change the placement of grid points, but will over all give similar results. Also, when comparing ϵ with the accepted error for linear interpolation, $\mathcal{O}(\frac{1}{N^2})$, it should be noted that ϵ will be on the order of $\frac{1}{N^2}$, but the constant will be improved.

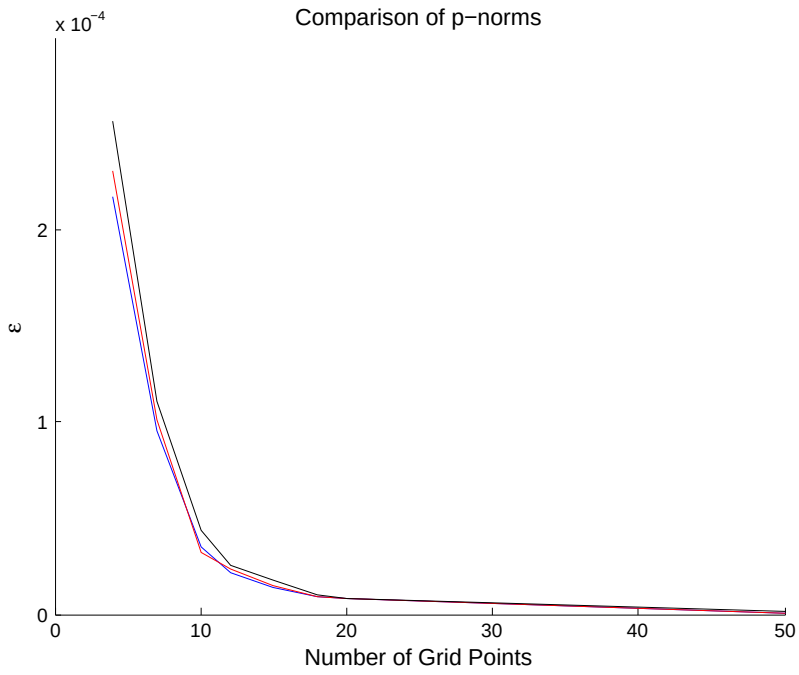


Figure 1: Comparison of 2-norm (blue), 3-norm (red) and 7-norm (black).

2 The Nelder-Mead Simplex Method

2.1 Nelder-Mead Simplex Method

The Nelder-Mead Simplex Method is an algorithm which uses a simplex–the generalization of a tetrahedral region of space to n dimensions, in order to minimize functions. Given a function ϕ, with dimension n, the algorithm will take a simplex S, of $n+1$ vertices and locate a minimum through a series of reflections, expansions, contractions and shrinkings. Because it relies only on function values at specific points, no derivatives are needed to locate minima as in traditional methods. The following will illustrate one iteration of the method [2][1].

2.1.1 Creating and Ordering the Simplex

For a function ϕ in n-dimensional space, $n+1$ vertices $\{G_1, G_2, \ldots, G_{n+1}\}$ shall be chosen to create the initial simplex. Once the vertices of the simplex have been selected, they shall be ordered and reassigned indices according to their function values when evaluated. Once ordered, the simplex $(G_1, G_2, \ldots, G_{n+1})$ corresponds to the fact that $\phi(G_1) \leq \phi(G_2) \leq \cdots \leq \phi(G_{n+1})$. Furthermore we define $\phi_h = \max(\phi_i)$, $\phi_s = \max_{i \neq h}(\phi_i)$, and $\phi_l = \min(\phi_i)$, where $\phi_i = \phi(G_i)$.

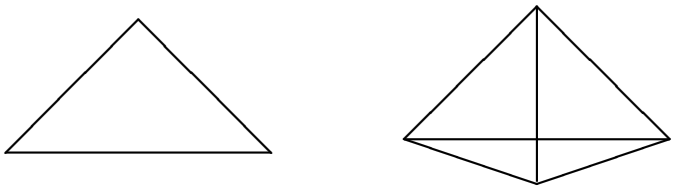


Figure 2: A 2-simplex and 3-simplex.

2.1.2 Calculating the Centroid of the Simplex

The centroid $\bar{G}$ of the 'best' side of the simplex (opposite the worst vertex) can now be calculated by excluding vertex G_h as follows:

$$\bar{G} = \frac{1}{n} \sum_{i \neq h} G_i$$

The centroid will be used in later steps to compute reflection, expansion, and contraction vertices.

2.1.3 Reflecting the Simplex

The reflection vertex G_r and its function value ϕ_r will be calculated as follows:

$$G_r = (1 + \alpha)\bar{G} - \alpha G_h$$

$$\phi_r = \phi(G_r)$$

where α is the reflection coefficient, $\alpha > 0$, with the standard value being $\alpha = 1$.

If $\phi_l \leq \phi_r < \phi_s$, G_r will replace ϕ_h, and the iteration will terminate. Otherwise, either an expansion or a contraction will be performed on the simplex.

2.1.4 Expanding the Simplex

The simplex will be expanded if $\phi_r < \phi_l$. The reasoning behind this is that if the reflection has brought the simplex close to the minimum of the function, an expansion might bring the simplex closer. The expansion vertex and its function value are calculated as follows:

$$G_e = \gamma G_r + (1 - \gamma)\bar{G}$$

$$\phi_e = \phi(G_e)$$

where γ is the expansion coeffiecient, $\gamma > 1$, with the standard value being $\gamma = 2$.

If $\phi_e < \phi_r$, replace G_h by G_e and terminate the iteration. Otherwise $\phi_e \geq \phi_r$, and G_r replaces G_h before terminating.

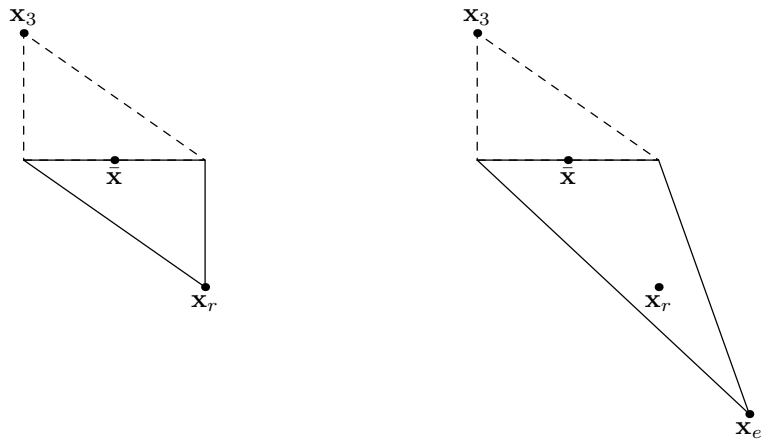


Figure 3: A simplex after the reflection step and after the expansion step with centroid $\bar{x}$, reflection vertex x_r and expansion vertex x_e. The original simplex is shown with dashed lines [1].

2.1.5 Contracting the Simplex

A contraction of the simplex will be performed if $\phi_r \geq \phi_s$. There are two types of contractions, an outside contraction and and inside contraction.

If $\phi_s \leq \phi_r < \phi_h$, an outside contraction will be peformed by calculating the contraction vertex G_c and its function value as follows:

$$G_c = \beta G_r + (1 - \beta)\bar{G}$$

$$\phi_c = \phi(G_c)$$

where β is the contraction coefficient, $0 < \beta < 1$, with the standard value being $\beta = \frac{1}{2}$.

If $\phi_c \leq \phi_r$, replace G_h by G_c and terminate the iteration. Otherwise, perform a shrinkage on the simplex.

If $\phi_r \geq \phi_h$, an inside contraction will be performed by calculating the contraction vertex:

$$G_c = \beta G_h + (1 - \beta)\bar{G}$$

$$\phi_c = \phi(G_c)$$

If $\phi_c < \phi_h$, replace G_h with G_c and terminate the iteration. Otherwise, perform a shrinkage on the simplex.

2.1.6 Shrinking the Simplex

In order to shrink the simplex, n new vertices will be created in addition to accepting G_l. These vertices are chosen as follows with shrinkage coefficient δ, $0 < \delta < 1$, with standard value $\delta = \frac{1}{2}$:

$$G_i = \delta G_i + (1 - \delta)G_l$$

$$\phi_i = \phi(G_i),$$

$i = 2, \ldots, n + 1$. Once the simplex has shrunk, the iteration is terminated.

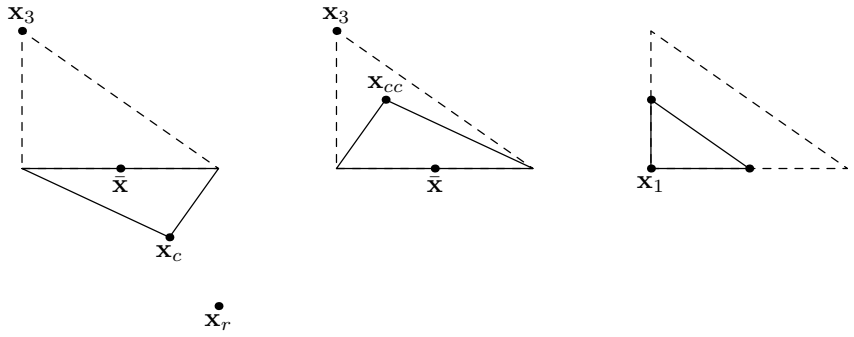


Figure 4: A simplex after an outside contraction, and inside contraction and a shrinkage with outside contraction vertexx_c, inside contraction vertex x_{cc} and best vertex x_l. The original simplex is shown with dashed lines [1].

2.1.7 Termination of the Nelder-Mead Simplex Method

The Nelder-Mead Simplex method can terminate due to one of three different criteria: convergence of the vertices, convergence of the function values, or failure.

Convergence of Vertices: Convergence of this type occurs when some or all of vertices G_i are sufficiently close together (i.e. $|G_i - G_j| \leq \epsilon, \forall i, j \in \{1, \ldots, n+1\}, i \neq j.$) This type of convergence is useful when applying the algorithm to discontinous functions [5].

Convergence of Function Values: Convergence occurs when some or all of the function values ϕ_i are sufficiently close (i.e. $|\phi_i - \phi_j| \leq \epsilon, \forall i, j \in \{1, \ldots, n+1\}, i \neq j.$) Most implementations use this convergence criteria when searching for a minimum. It is also useful for finding an accurate approximation when an exact minimum cannot be reached, such as with

'noisy' functions [5].

Failure: Occurs when the number of iterations exceeds an allowed amount with little improvement being made to minimizing function values between iterations [5].

2.1.8 Comments about the Method

As with any method, there are specific advantages and disadvantages to using the Nelder-Mead Simplex Method.

Advantages:

- Compared with other minimization methods, the Nelder-Mead method is fast, requiring one function evaluation when the iteration terminates at the reflection step, two function evaluations when the iteration terminates at the expansion or contraction step, and n+2 function evaluations when the iteration terminates at the shrinkage step [1].
- When full optimization cannot occur and an exact minimum cannot be reached, the Nelder-Mead method can make resonably accurate approximations, reaching values arbitrarily close to the minimum.

Disadvantages:

- Restricted domains of functions may cause problems in locating a minimum, such as premature termination. There are, however, modifications that may correct these problems such as scaling the domain or modifying the function [2].

- For a function with more than one minimum, the placement of the initial simplex may affect to which of the minima the algorithm converges. However, this convergence may give no insight on the nature of the function(i.e. the existence of minima.)

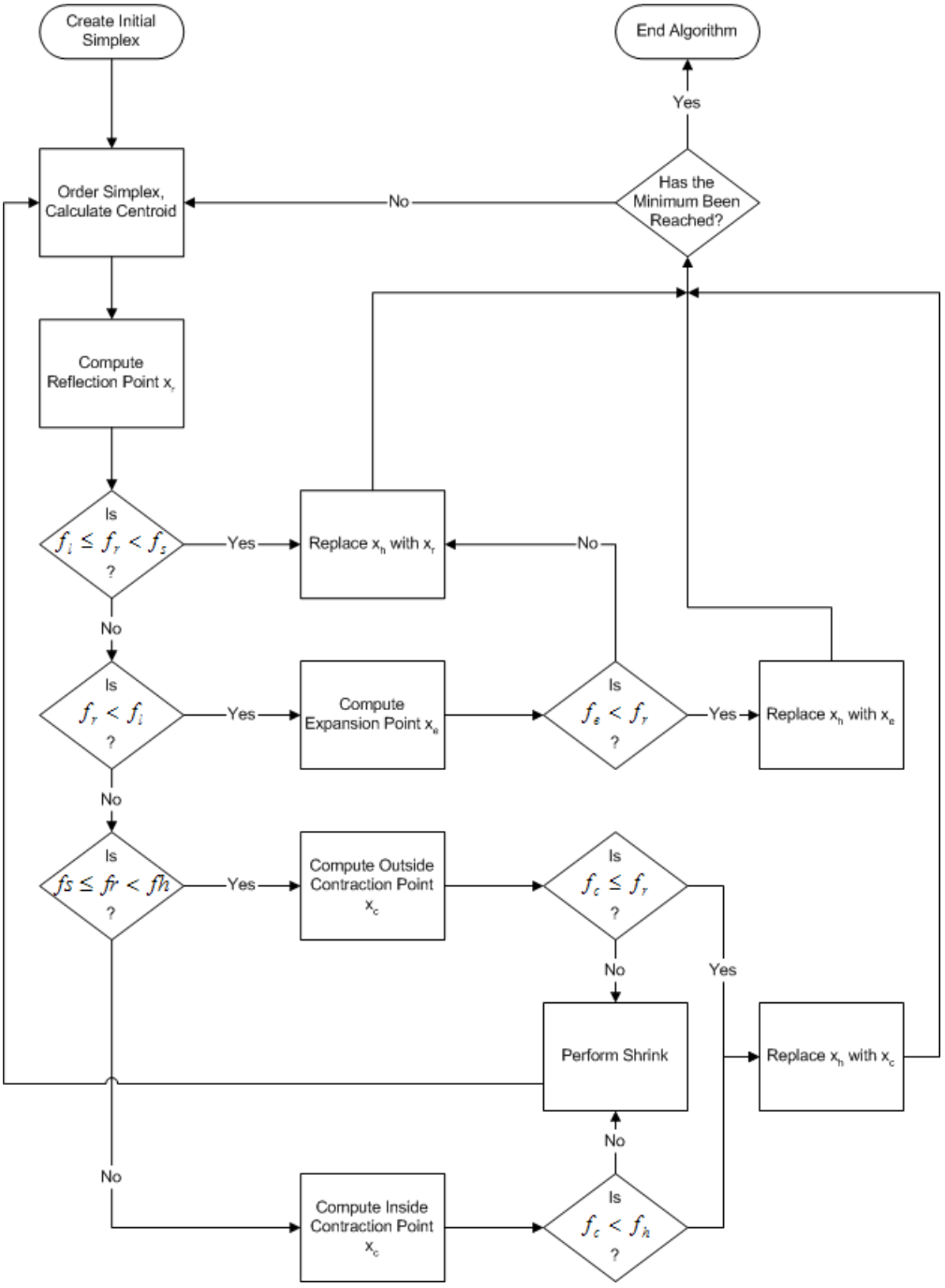


Figure 5: The Nelder-Mead Simplex Method

2.2 Modified Nelder-Mead Simplex Method for Interpolation Grids

In tailoring the Nelder-Mead Simplex Method to find interpolation grids that will minimize error, adjustments need to be made.

2.2.1 Creating and Ordering the Simplex

In order to use the Nelder-Mead Simplex method with interpolation grids, the definition of the simplex's vertices needs to be altered. In our case, each vertex now represents an interpolation grid G such that $G = (x_1, x_2, \ldots, x_N)$, where $x_1 < x_2 < \cdots < x_N$.

The Nelder-Mead Simplex Method requires $n + 1$ vertices to create the simplex S, defined as $S = \{G_1, G_2, \ldots, G_{N+1}\}$. In matrix form, simplex S can be represented as

$$S = \begin{bmatrix} G_1 \\ G_2 \\ \vdots \\ G_{N+1} \end{bmatrix}$$

and when expanded

$$S = \begin{bmatrix} x_{1,1} & x_{1,2} & \cdots & x_{1,N} \\ x_{2,1} & x_{2,2} & \cdots & x_{2,N} \\ \vdots & \vdots & \ddots & \vdots \\ x_{N+1,1} & x_{N+1,2} & \cdots & x_{N+1,N} \end{bmatrix}$$

In the above matrix, each row represents an interpolation grid, and $G_{i,j}$ is the j^{th} grid point of the i^{th} grid. Finally, since for each grid $x_1 = 0$ and $x_n = 1$, we have

$$S = \begin{bmatrix} 0 & x_{1,2} & \cdots & x_{1,N-1} & 1 \\ 0 & x_{2,2} & \cdots & x_{2,N-1} & 1 \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & x_{N+1,2} & \cdots & x_{N+1,N-1} & 1 \end{bmatrix}$$

where $0 < x_{i,j} < 1$.

Now, having ordered the grids such that $\epsilon(G_1) < \epsilon(G_2) < \cdots < \epsilon(G_{N+1})$, we can define $x_h = G_{N+1}$, $\epsilon_h = \epsilon(G_{N+1})$, $x_s = G_N$, $\epsilon_s = \epsilon(G_N)$, $x_l = G_1$, and $\epsilon_l = \epsilon(G_1)$, where ϵ is the error.

2.2.2 Calculating the Centroid

The centroid of interpolation grids $\bar{G}$ can be calculated by computing the arithmetic mean of each grid, excluding the grid that results in the largest error:

$$\begin{aligned} \bar{G} &= \frac{1}{N} \sum_{i=1}^{N} (x_{i,1}, x_{i,2}, \ldots, x_{i,N}) \\ &= \frac{1}{N} \Big(\sum_{i=1}^{N} x_{i,1}, \sum_{i=1}^{N} x_{i,2}, \ldots, \sum_{i=1}^{N} x_{i,n} \Big) \\ &= (\bar{x}_1, \bar{x}_2, \ldots, \bar{x}_N) \end{aligned}$$

Geometrically, $\bar{G}$ will be located across from $G(N+1)$.

2.2.3 Simplex Transformations

The reflection grid, expansion grid, contraction grid, and shrinkage grids are calculated as follows:

Reflection Grid

The reflection grid points are computed as follows, with $\alpha = 1$.

$$x_{r,i} = \bar{x}_i + \alpha(\bar{x}_i - x_{N+1,i})$$

for $1 \leq i \leq N$, giving the resulting reflection grid:

$$\begin{aligned} G_r &= (x_{r,1}, x_{r,2}, \ldots, x_{r,N}) \\ &= (x_{r1}, x_{r2}, \ldots, x_{rN}) \end{aligned}$$

Expansion Grid

The expansion grid points are computed as follows, with $\gamma = 2$.

$$x_{e,i} = \bar{x}_i + \gamma(x_{ri} - \bar{x}_i)$$

for $1 \leq i \leq N$, giving the resulting expansion grid:

$$\begin{aligned} G_e &= (x_{e,1}, x_{e,2}, \ldots, x_{e,N}) \\ &= (x_{e1}, x_{e2}, \ldots, x_{eN}) \end{aligned}$$

Contraction Grid

The *outside* contraction grid points are computed as follows, with $\beta = \frac{1}{2}$.

$$x_{c,i} = \bar{x}_i + \beta(x_{ri} - \bar{x}_i)$$

for $1 \leq i \leq N$.

The *inside* contraction gridpoints are computed as follows, still with $\beta = \frac{1}{2}$ and $1 \leq i \leq N$.

$$x_{c,i} = \bar{x}_i + \beta(x_{i,N+1} - \bar{x}_i)$$

The resulting contraction grids from either contraction is

$$\begin{aligned} G_c &= (x_{c,1}, x_{c,2}, \ldots, x_{c,N}) \\ &= (x_{c1}, x_{c2}, \ldots, x_{cN}) \end{aligned}$$

Shrinkage Grids

To shrink the simplex and create N new grids, the following calculations are made, with $\delta = \frac{1}{2}$.

$$x_{i,j} = x_{1,j} + \delta(x^*_{i,j} - x_{1,j})$$

for $2 \leq i \leq N+1$, and $2 \leq j \leq N-1$. Here, $x^*_{i,j}$ is the grid point from the current iteration.

2.2.4 Comments about the Modified Method

For the Modified Nelder-Mead Simplex Method for Interpolation Grids, the initial simplex S is created by generating $N-2$ random numbers per grid between zero and one, and ordering them such that $0 < x_1 < x_2 < \cdots < x_{N-1} < 1$.

During some iterations, specifically in the reflection or expansion steps, simplex S can transform out of the domain in which it is restricted(i.e. $x_i > x_j$ when $i < j$.) This however is self-correcting in the sense that the error produced from this type of grid when interpolated will be greater than those grids that stay in the domain, and the simplex will eventually transform back into the proper domain.

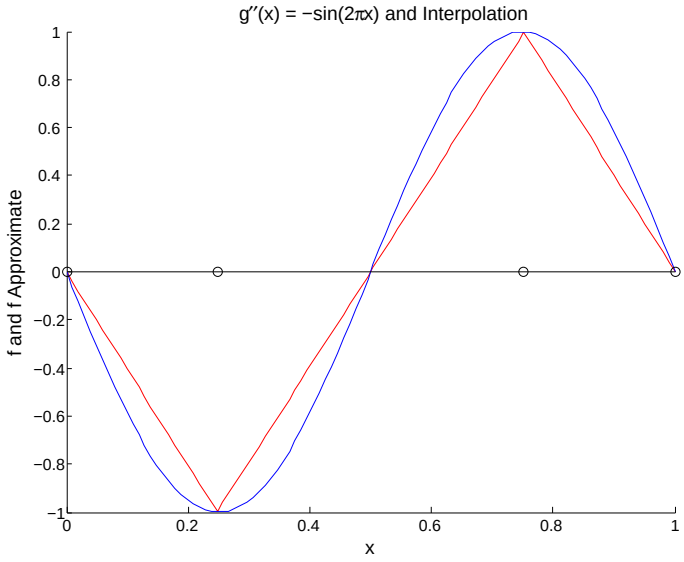
3 Results with Non-Normalized Errors

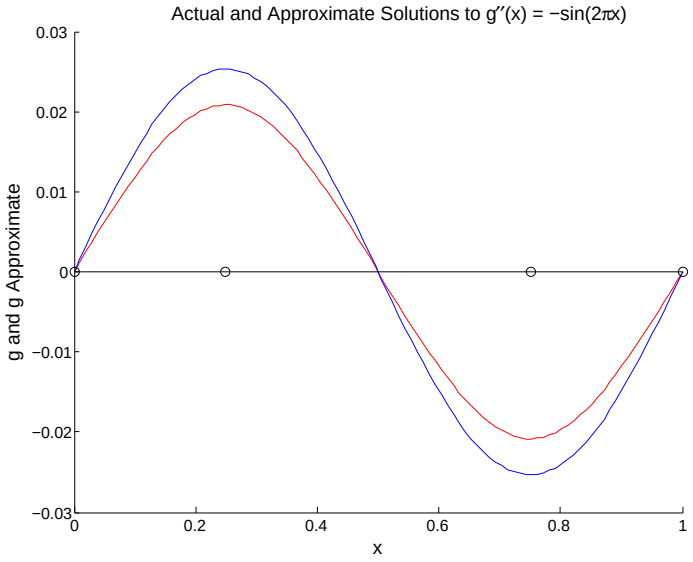
3.1 $g''(x) = -\sin(2\pi x)$

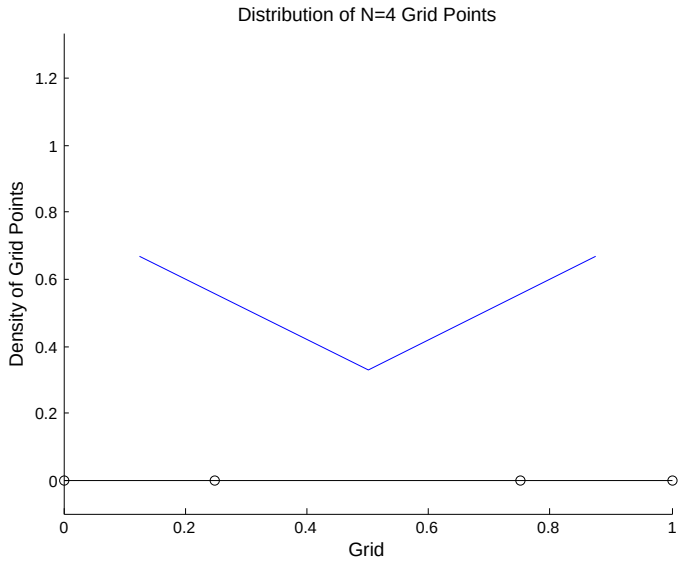
Solution $g(x) = \frac{\sin(2\pi x)}{4\pi^2}$

4 Grid Points

Error $3.39E-003$

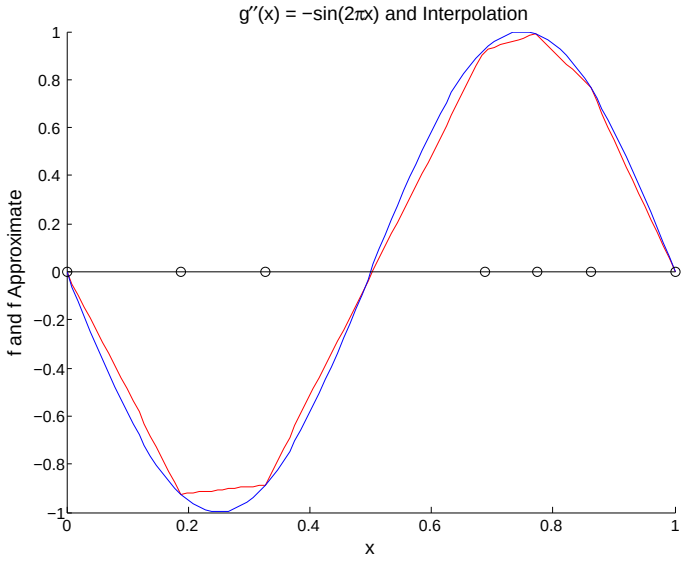


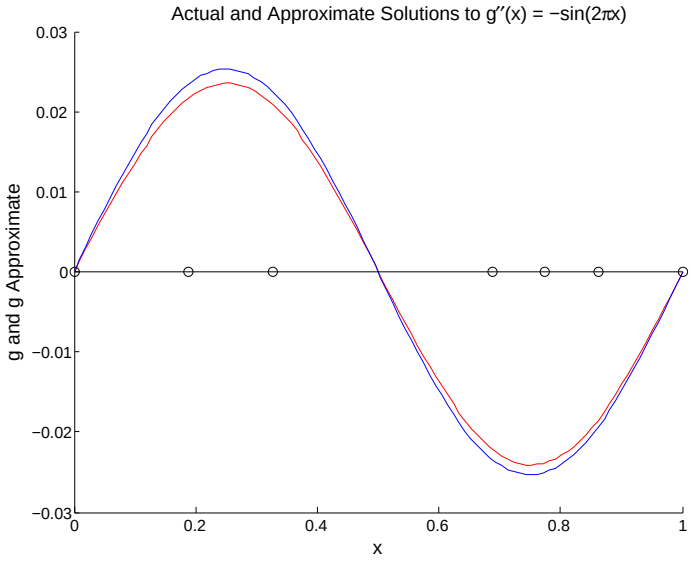


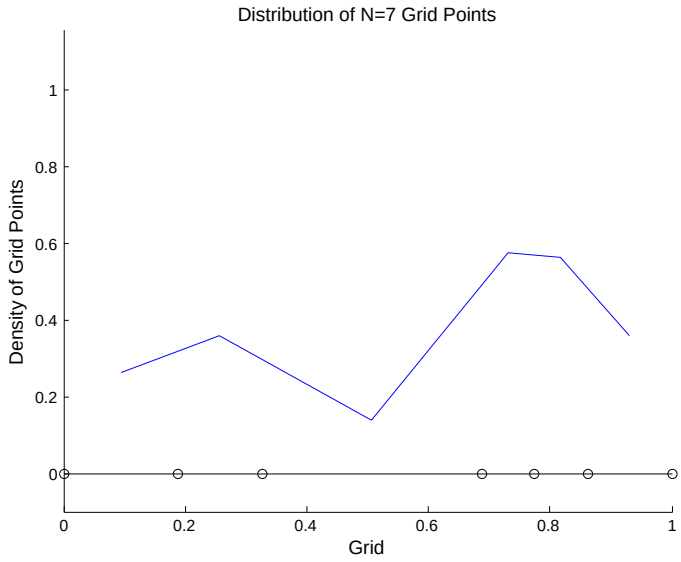


7 Grid Points

Error $1.15E-003$

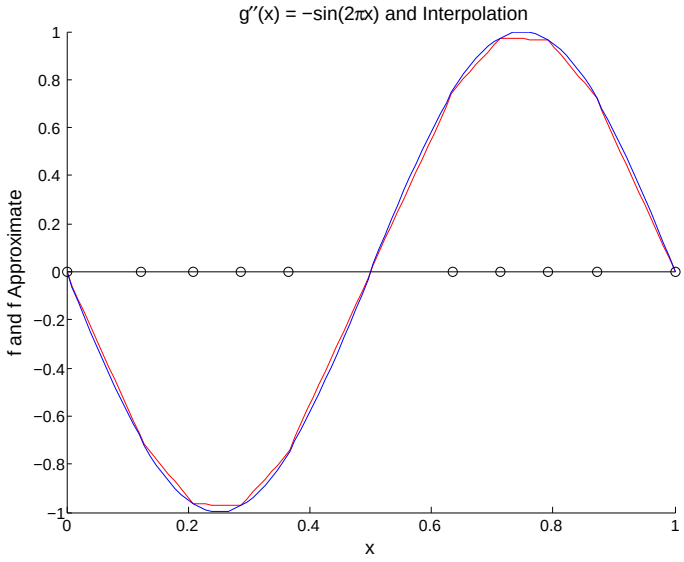


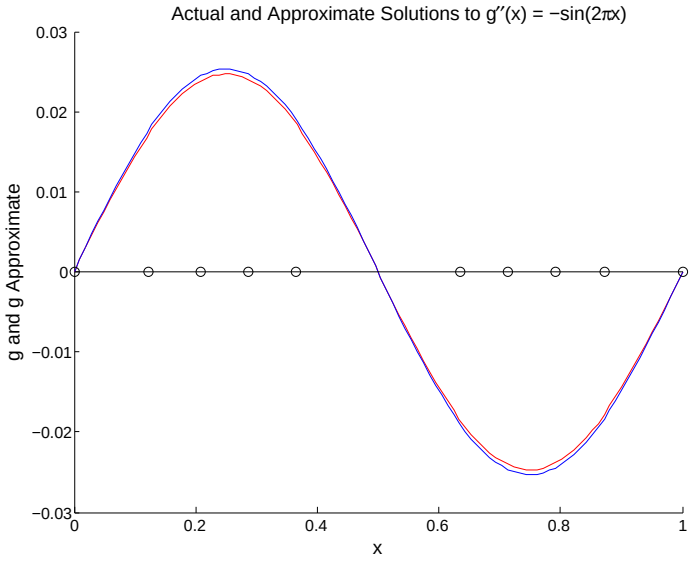


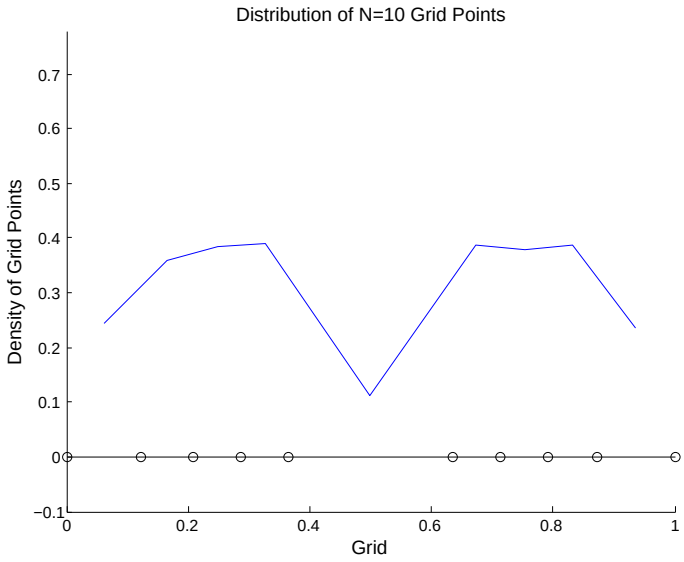


10 Grid Points

Error $4.63E-004$

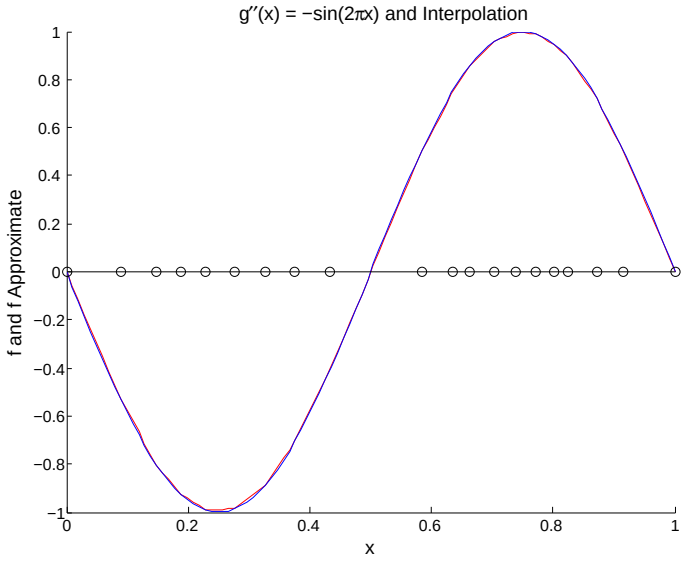


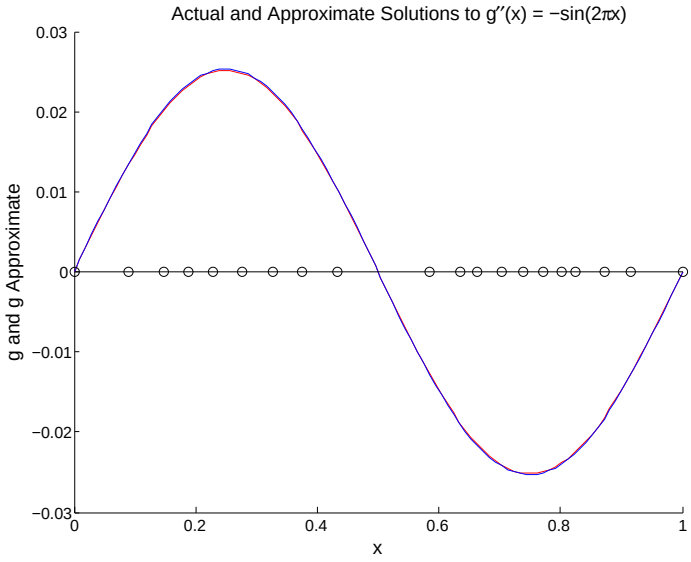


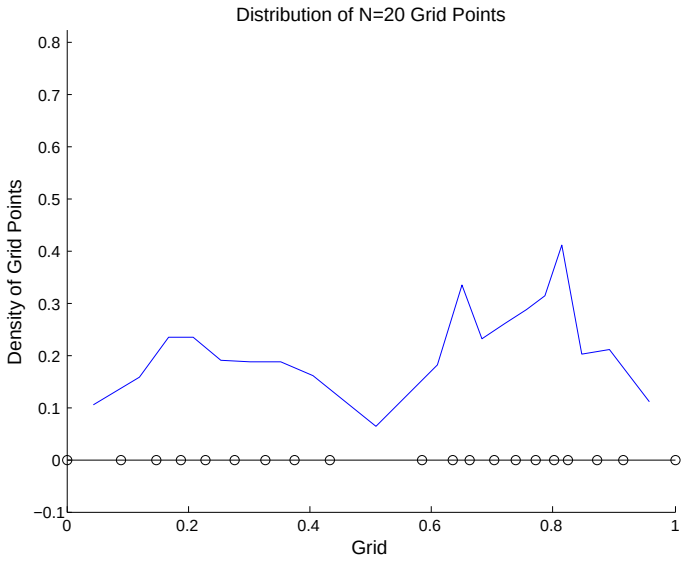


20 Grid Points

Error $1.17E-004$





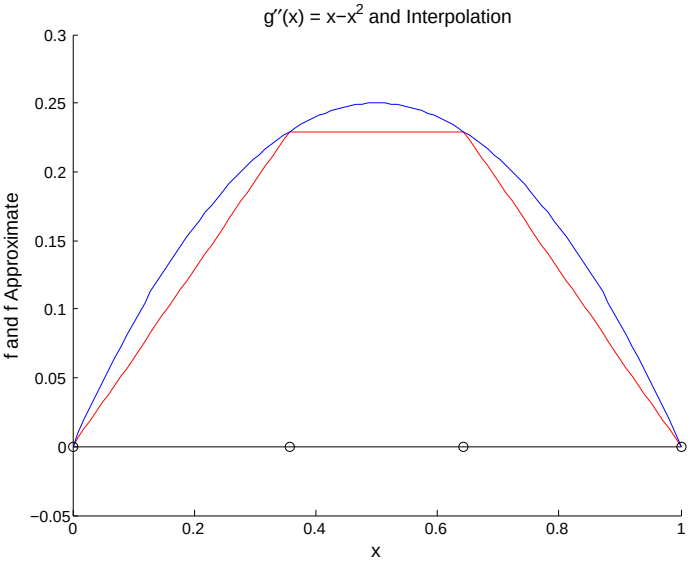


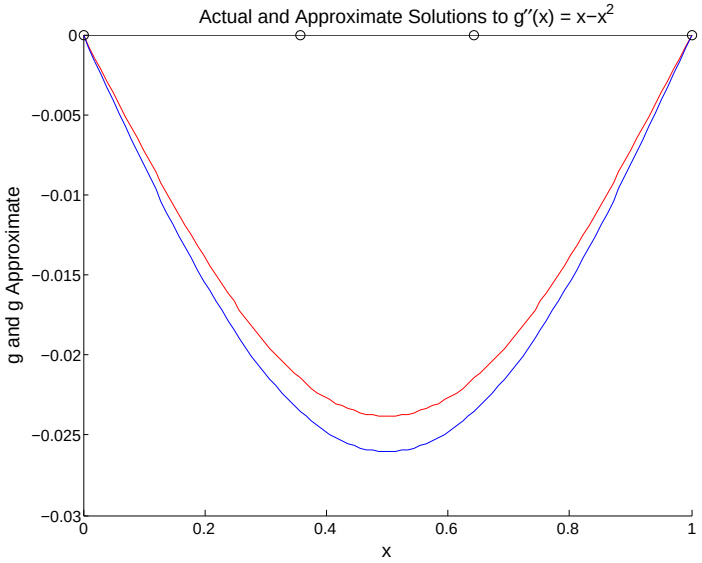
3.2 $g''(x) = x - x^2$

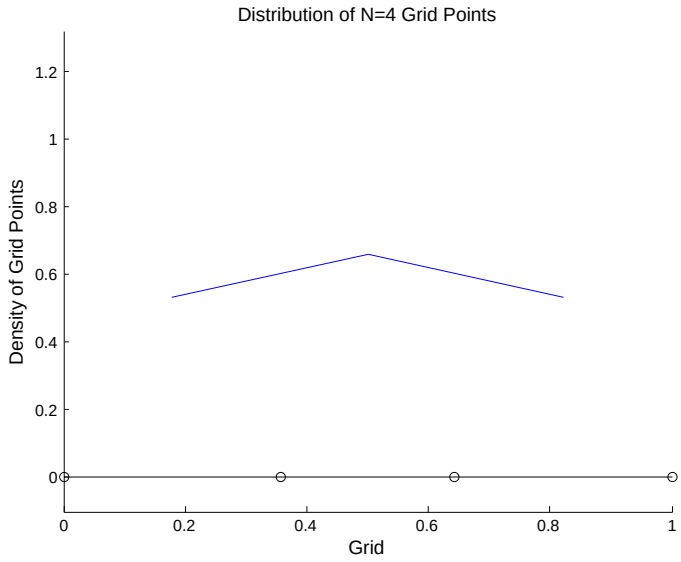
Solution $g(x) = \frac{x^3}{6} - \frac{x^4}{12} - \frac{x}{12}$

4 Grid Points

Error $1.67E-003$

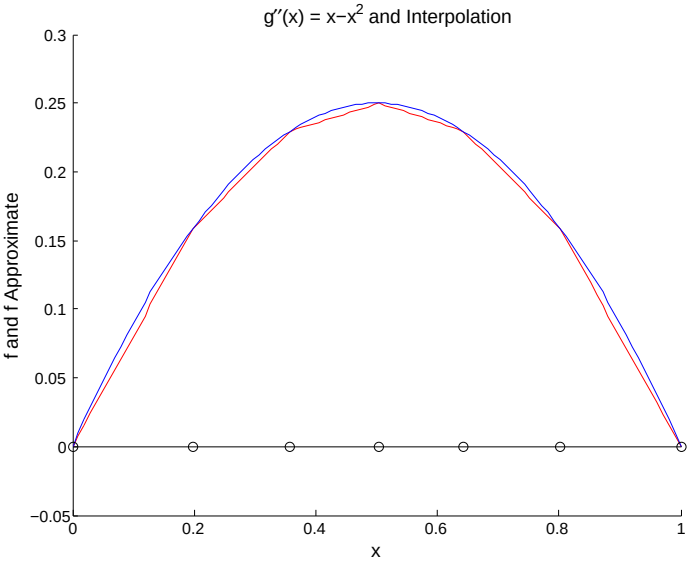


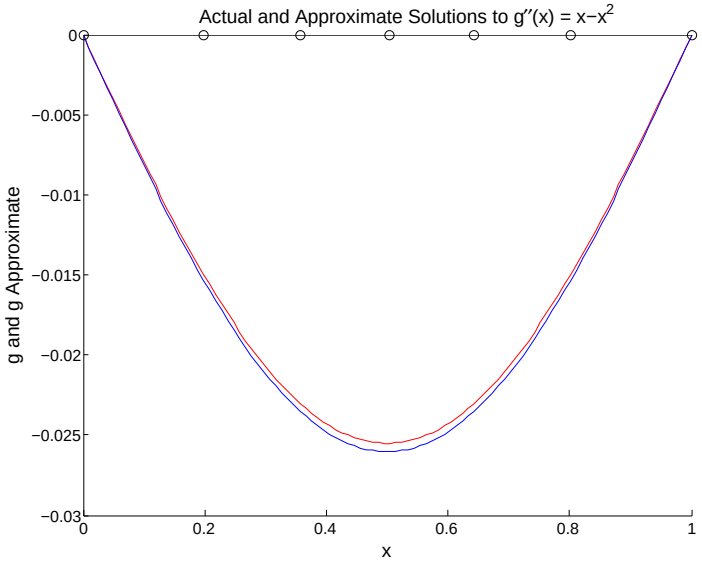


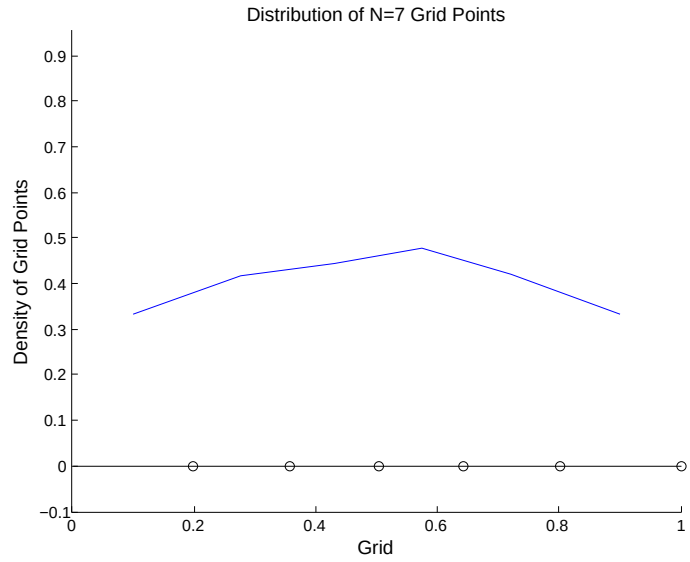


7 Grid Points

Error $3.93E-004$

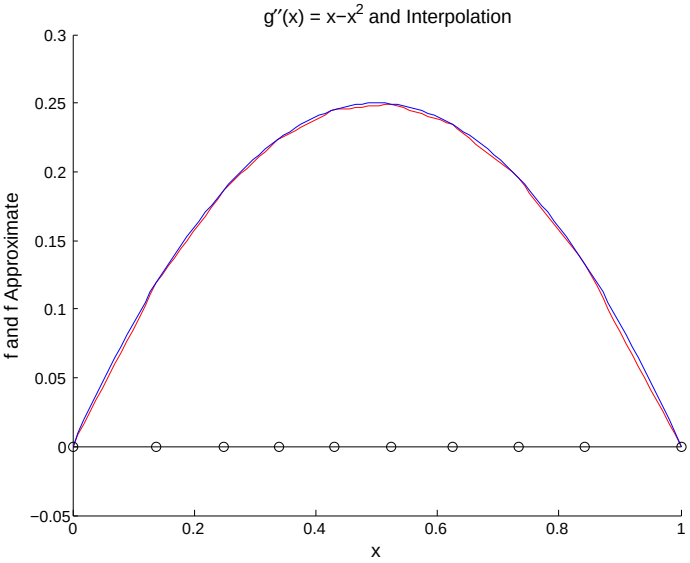


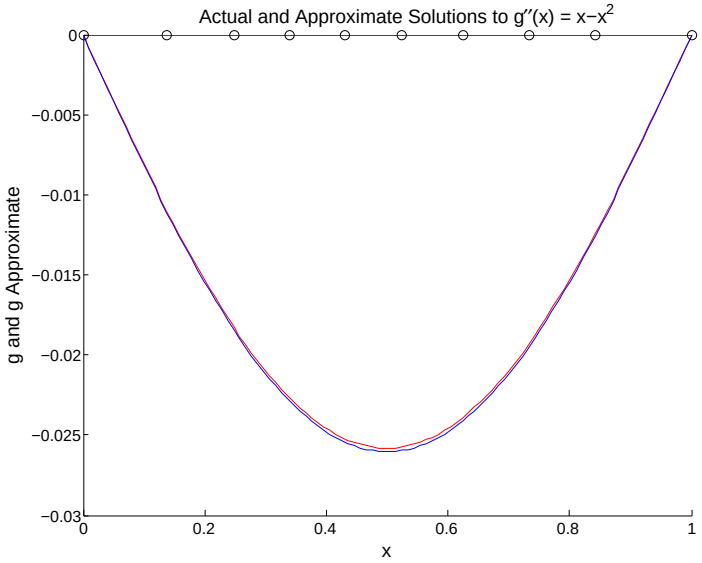


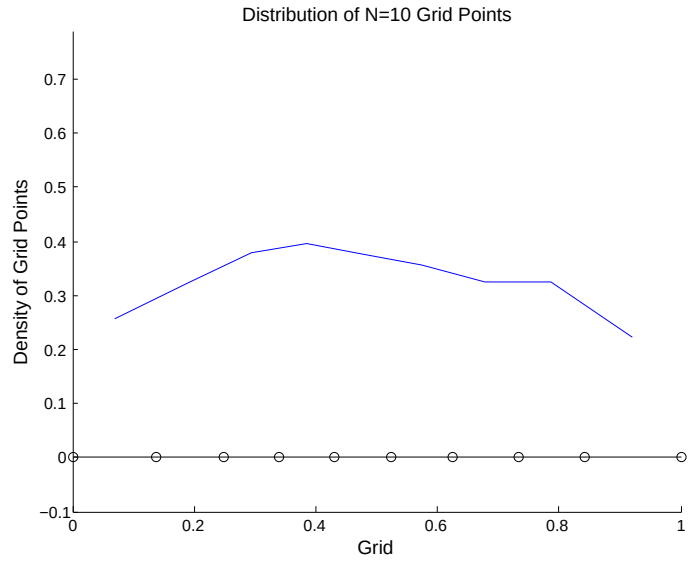


10 Grid Points

Error $1.72E-004$

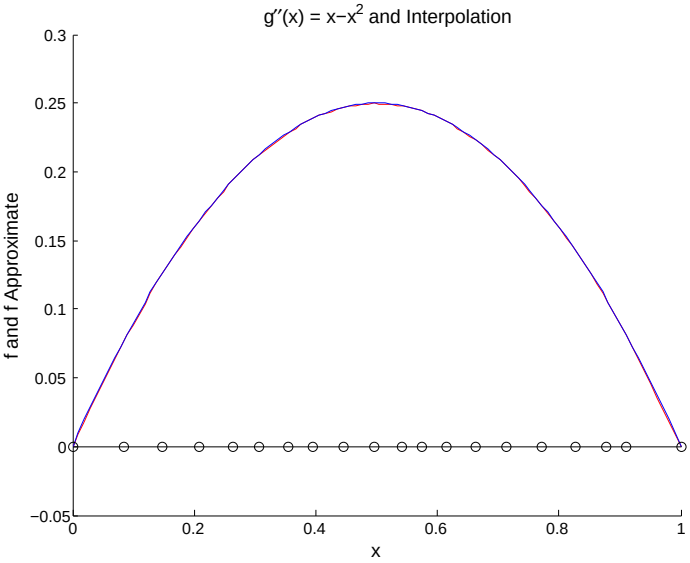


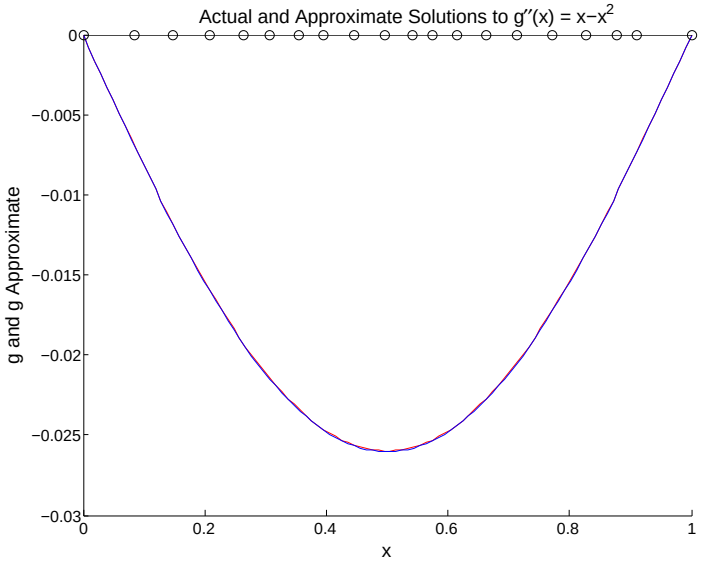


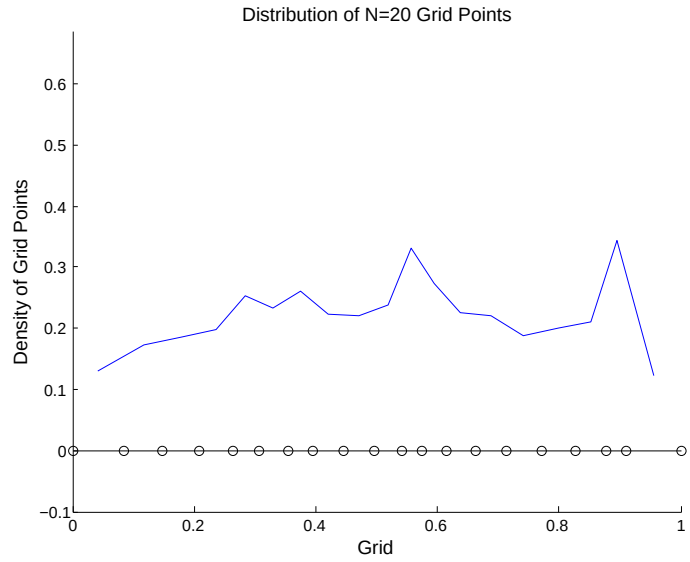


20 Grid Points

Error $3.80E-005$





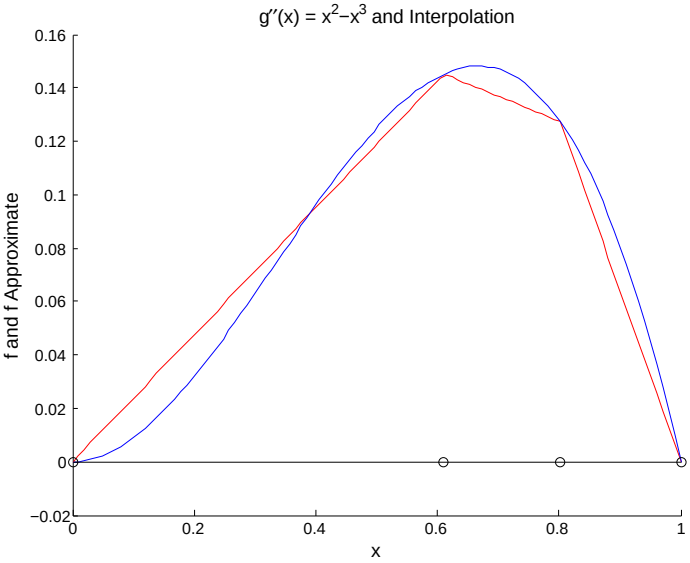


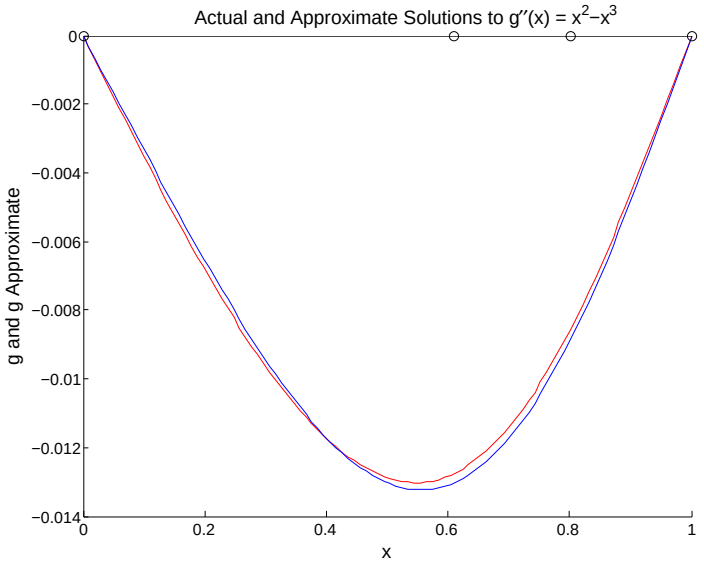
3.3 $g''(x) = x^2 - x^3$

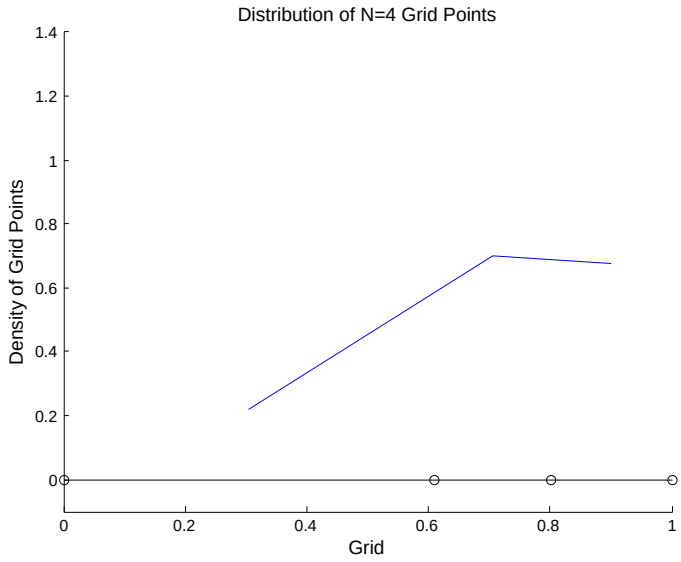
Solution $g(x) = \frac{x^4}{12} - \frac{x^5}{20} - \frac{x}{30}$

4 Grid Points

Error $2.17E-004$

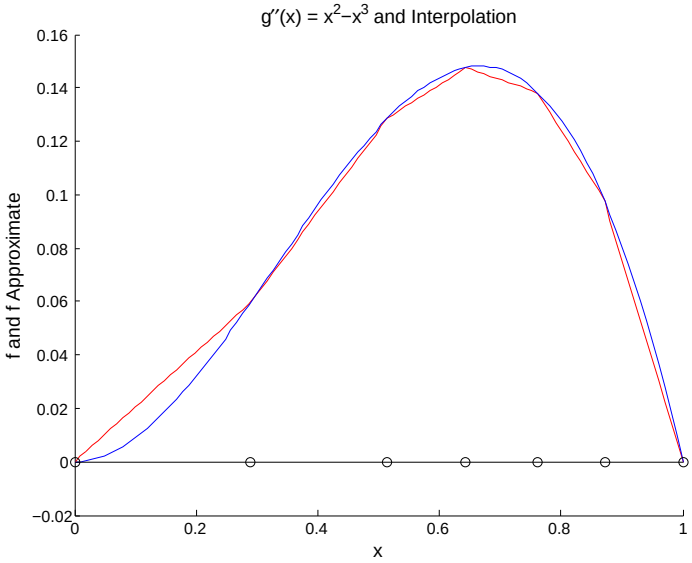


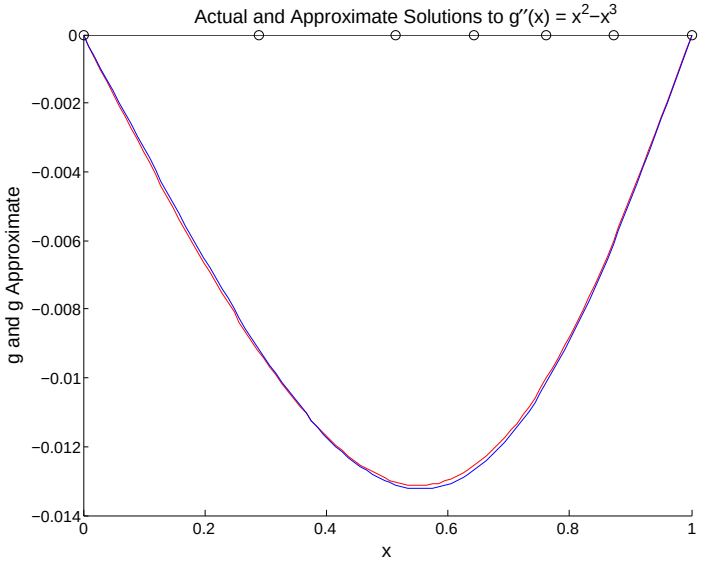


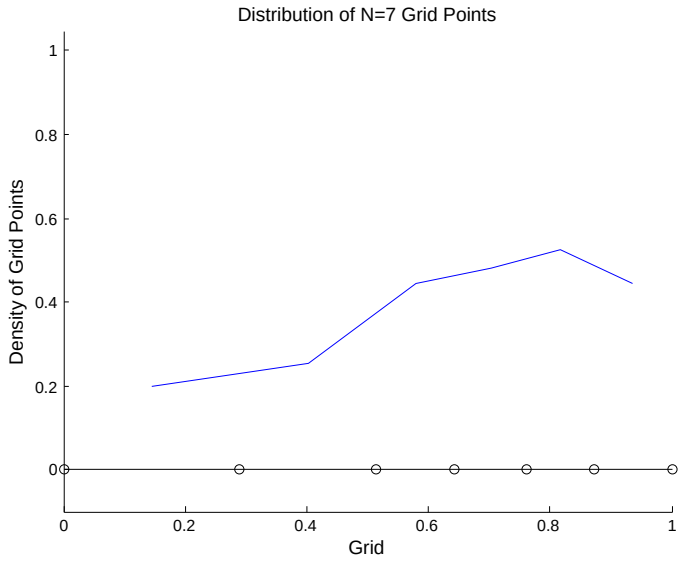


7 Grid Points

Error $9.50E-005$

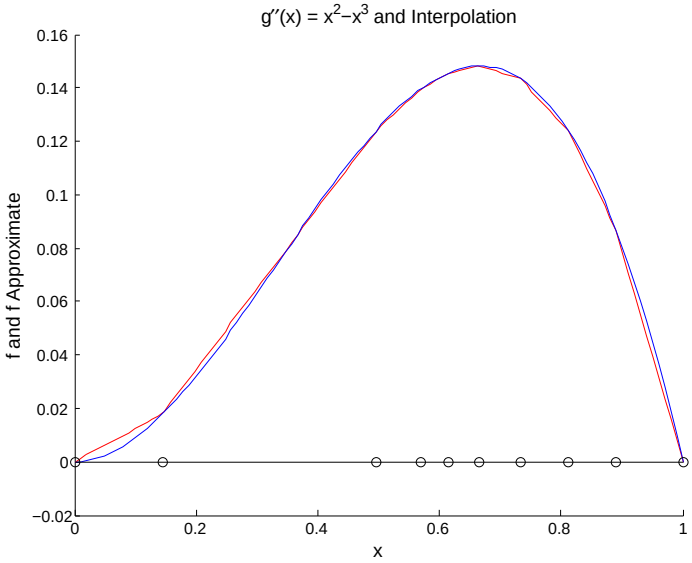


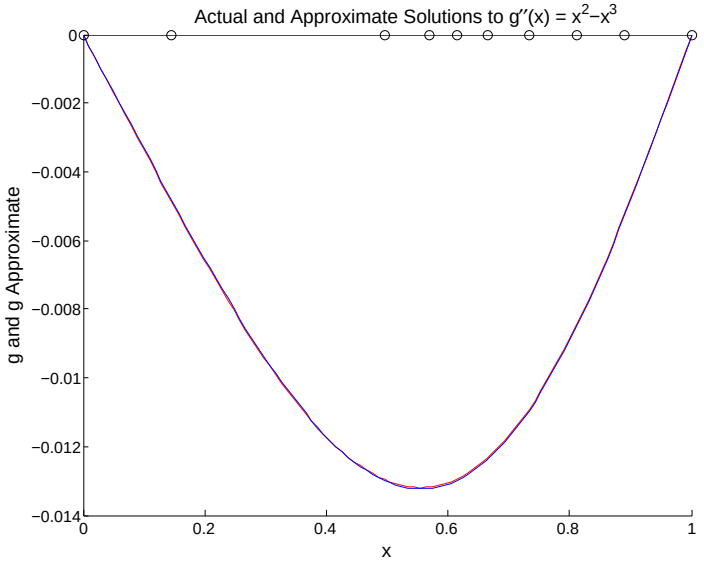


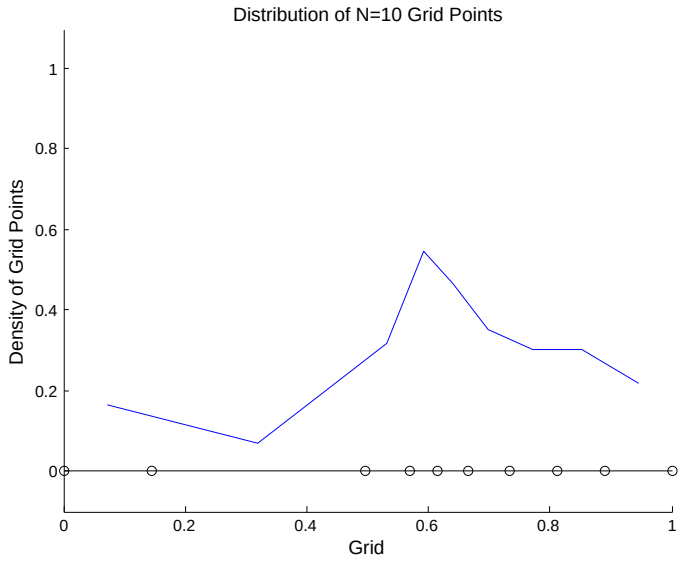


10 Grid Points

Error $3.45E-005$

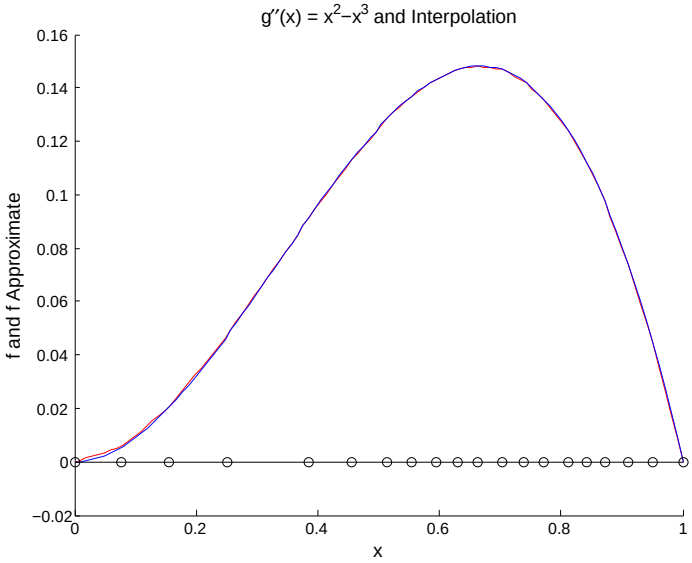


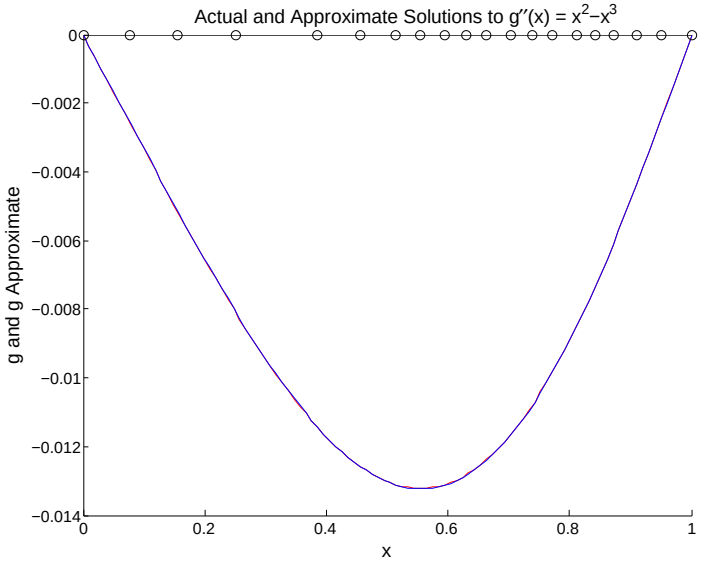


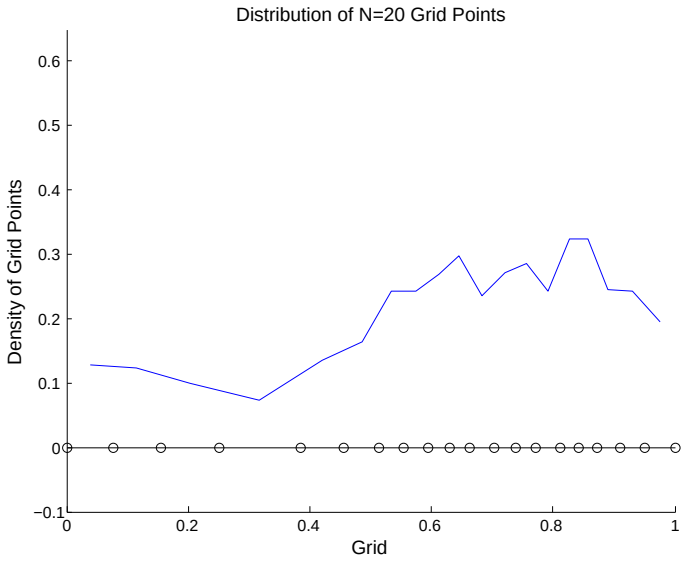


20 Grid Points

Error $7.76E-006$





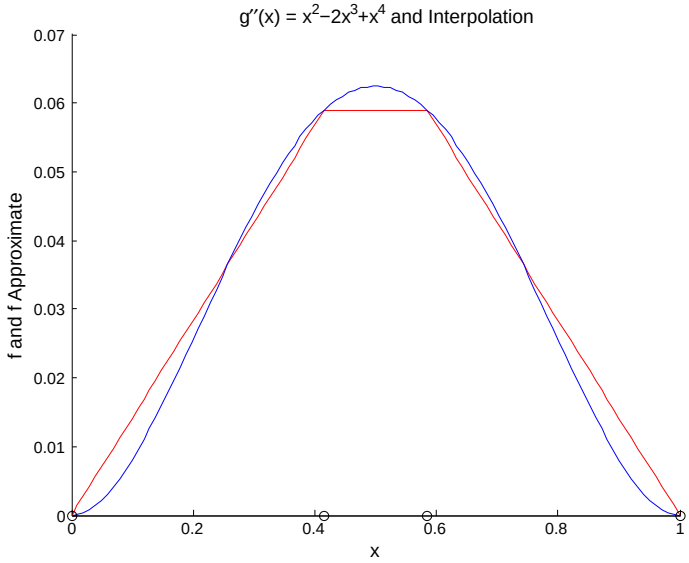


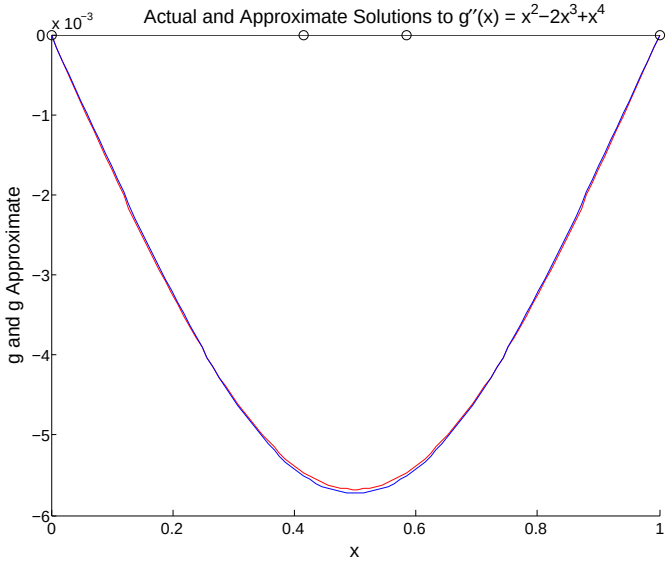
3.4 $g''(x) = x^2(1-x)^2$

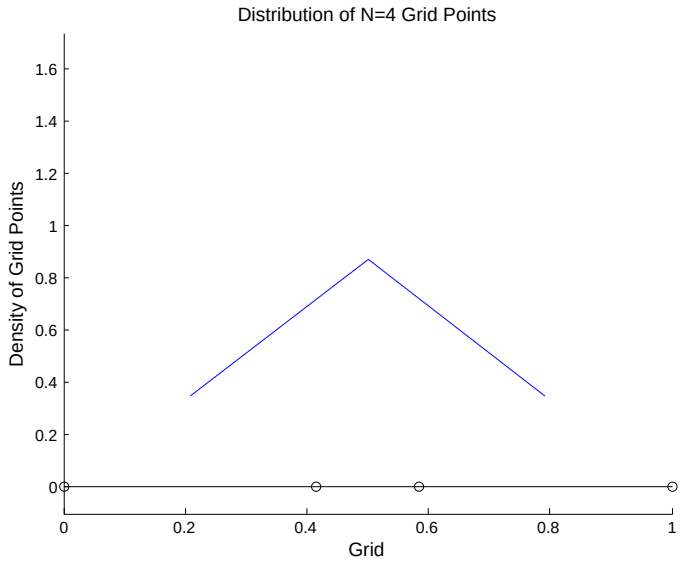
Solution $g(x) = \frac{x^4}{12} - \frac{x^5}{10} - \frac{x^6}{30} - \frac{x}{60}$

4 Grid Points

Error $3.33E-005$

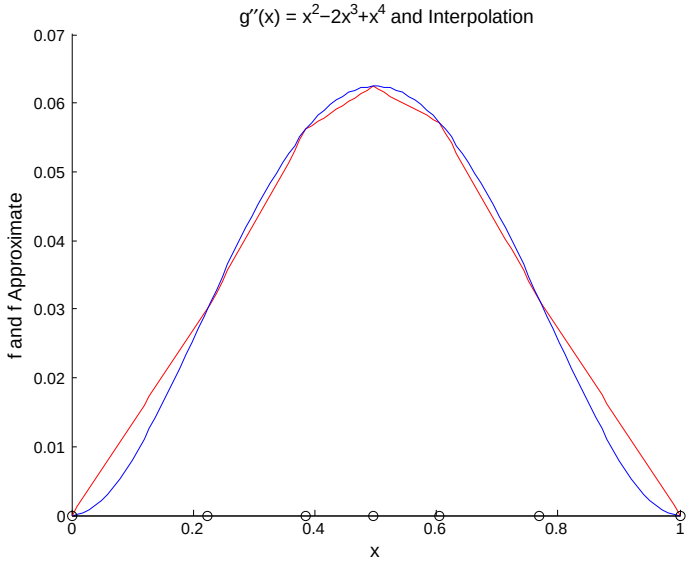


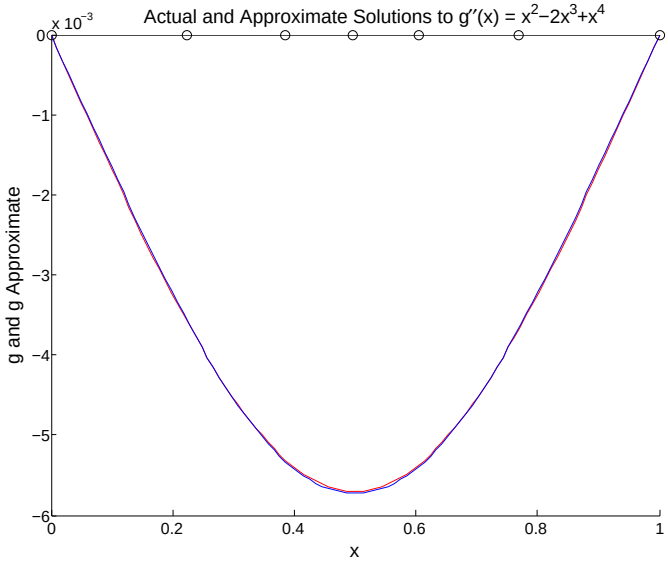


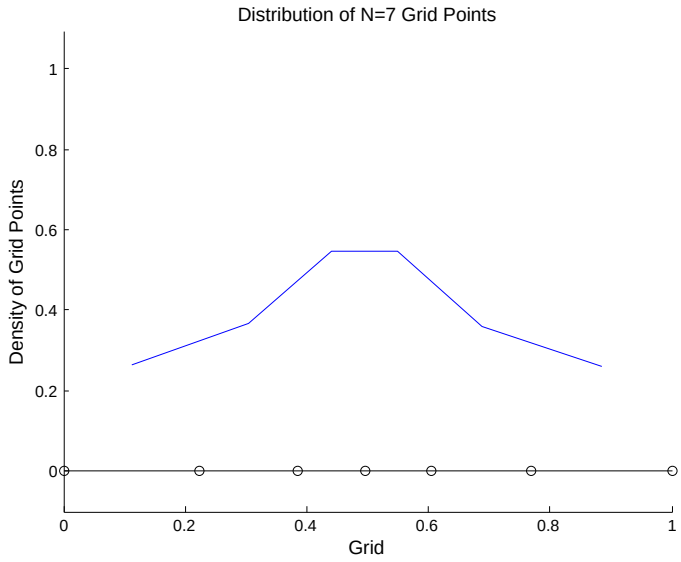


7 Grid Points

Error $2.22E-005$

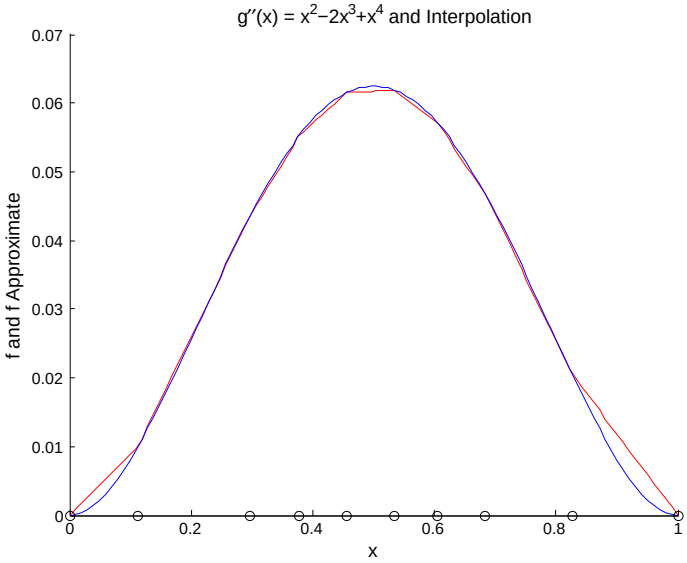


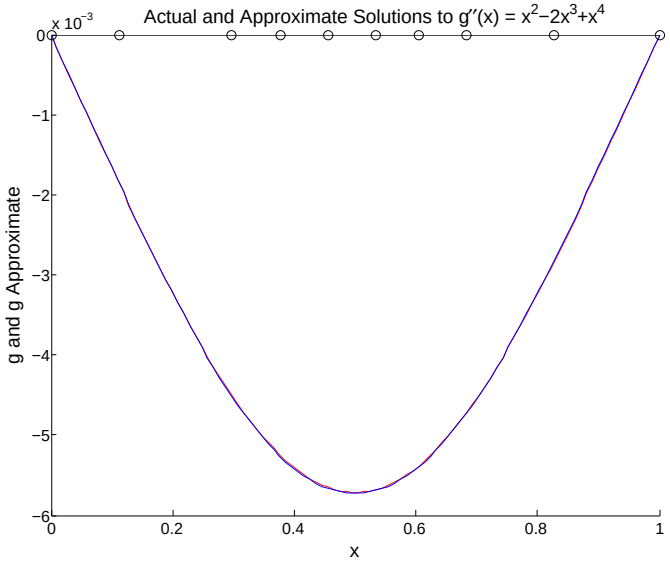


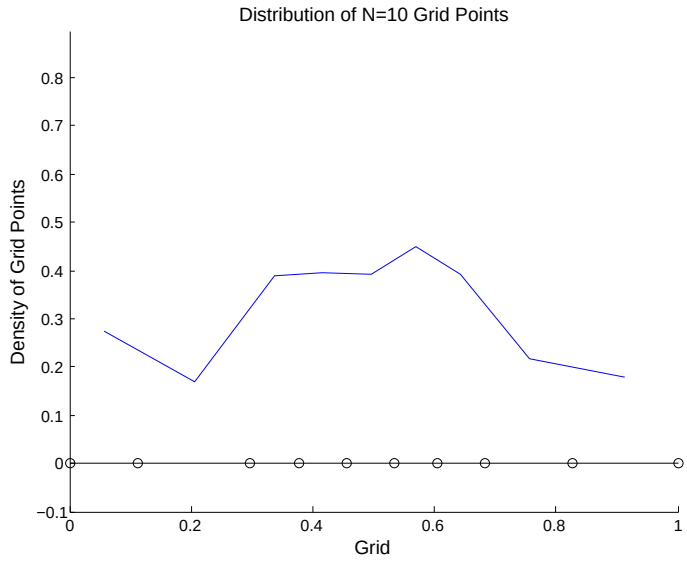


10 Grid Points

Error $9.97E-006$

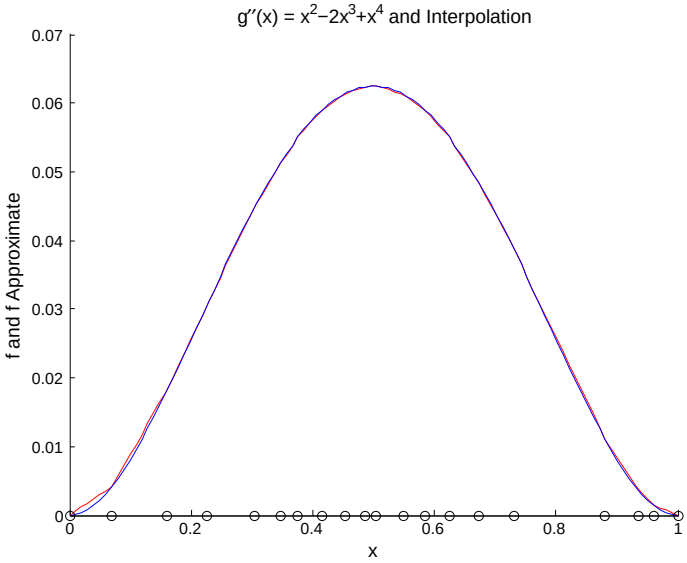


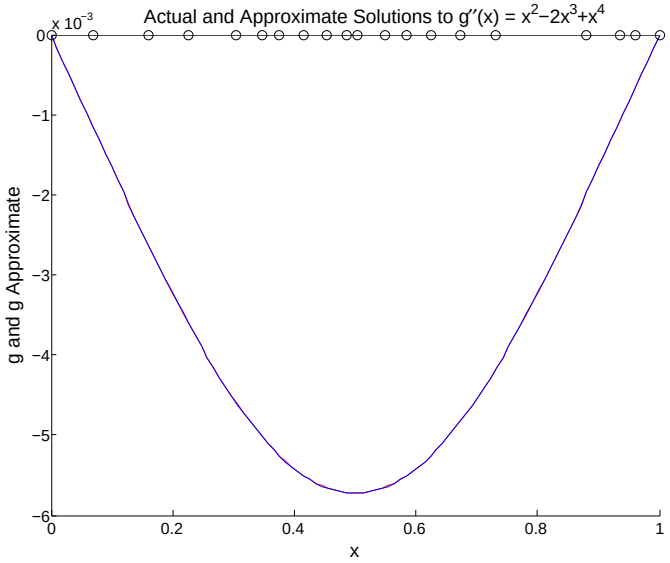


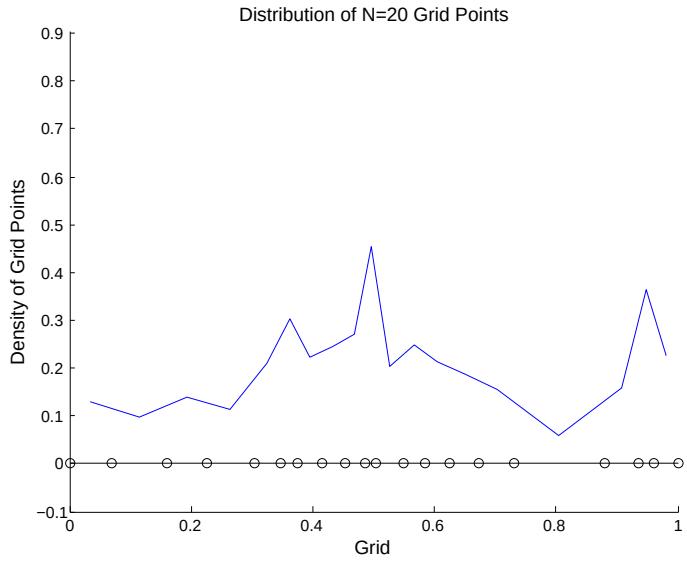


20 Grid Points

Error $2.05E-006$





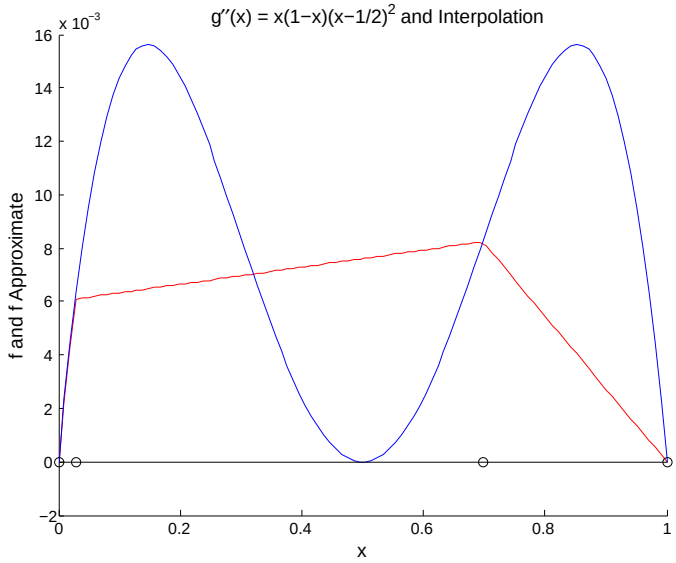


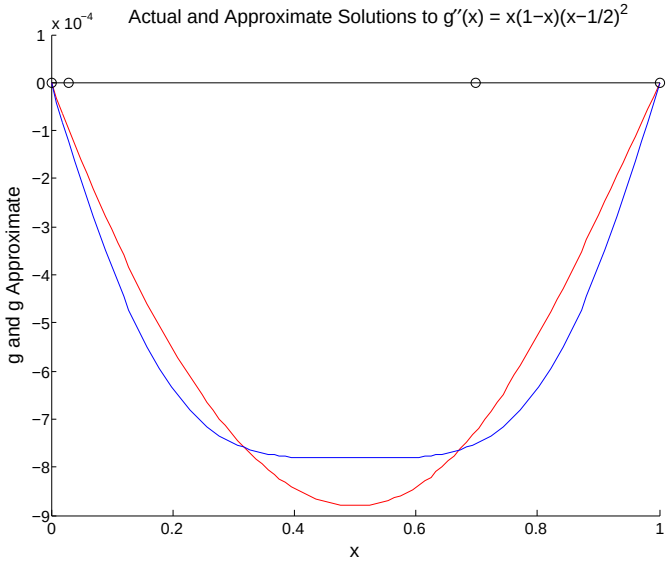
3.5 $g''(x) = x(1-x)(x-\frac{1}{2})^2$

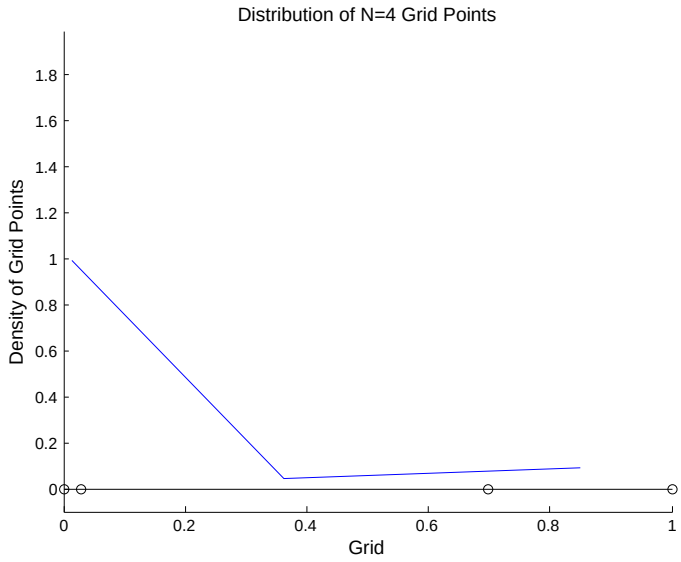
Solution $g(x) = \frac{x^5}{10} - \frac{x^6}{30} - \frac{5x^4}{48} + \frac{x^3}{24} - \frac{x}{240}$

4 Grid Points

Error $7.41E-005$

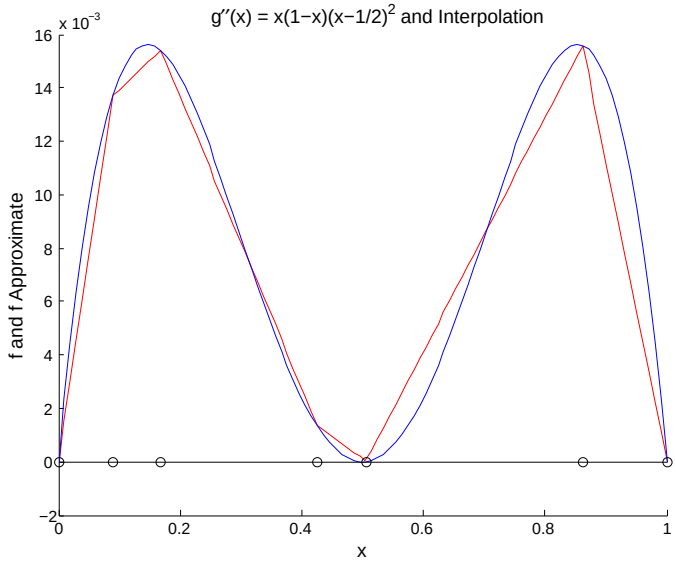


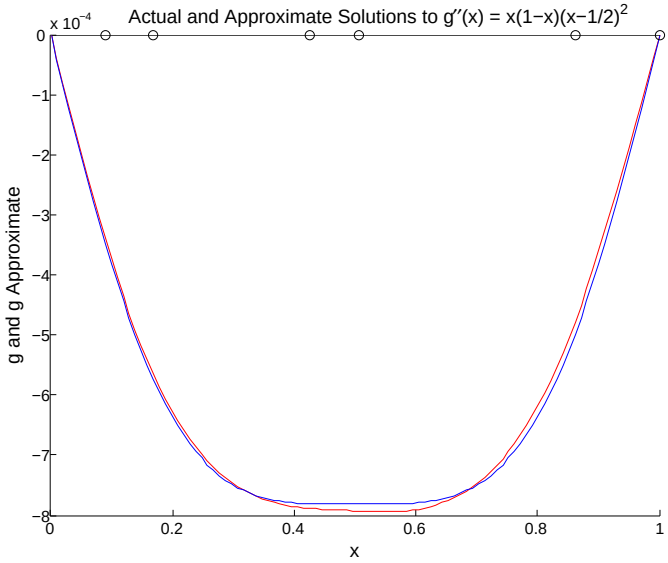


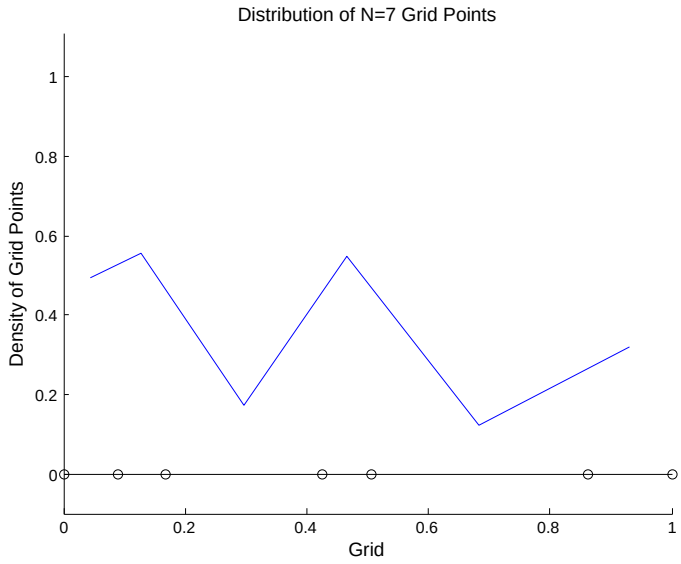


7 Grid Points

Error $1.12E-005$

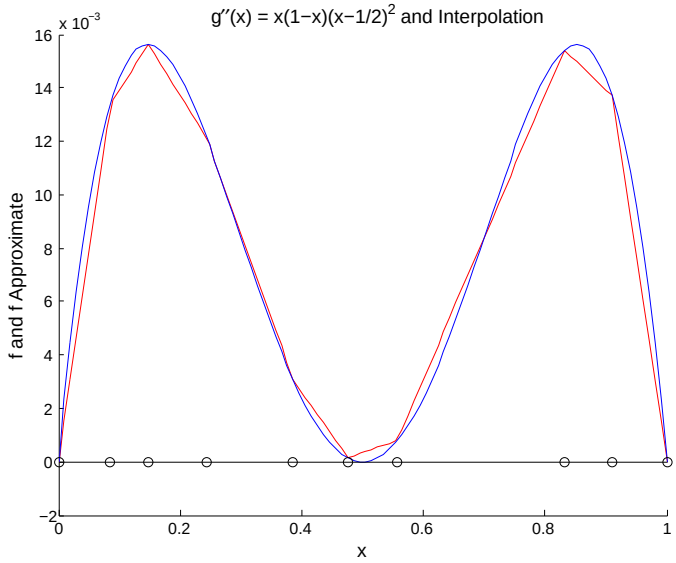


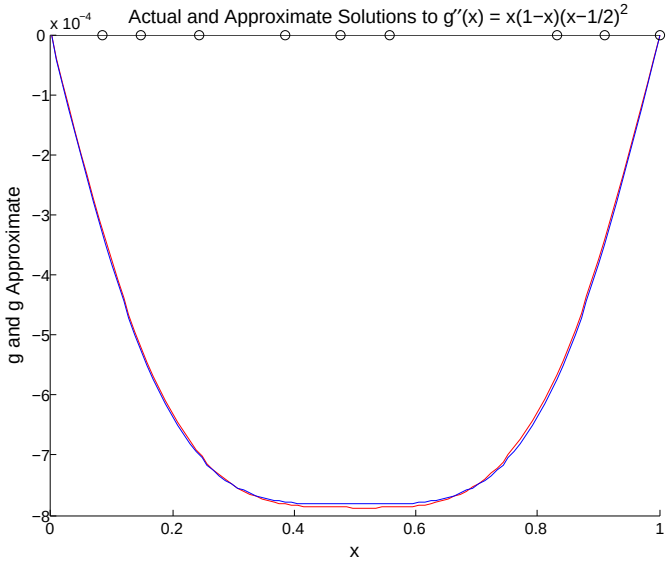


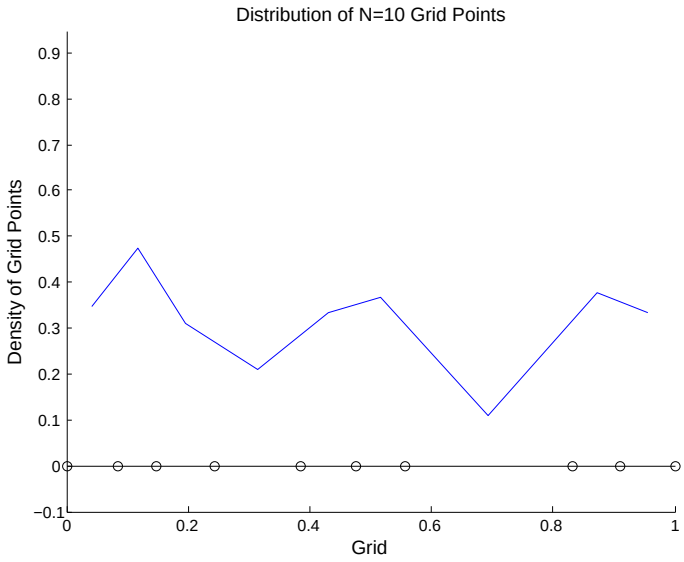


10 Grid Points

Error $5.19E-006$

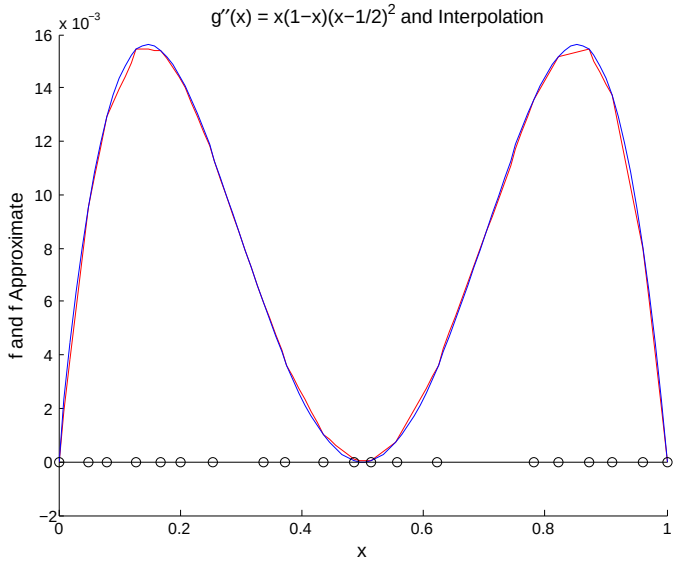


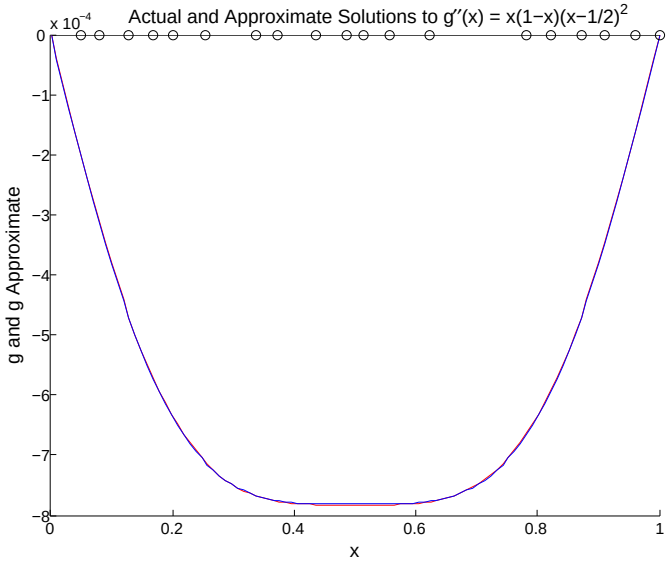


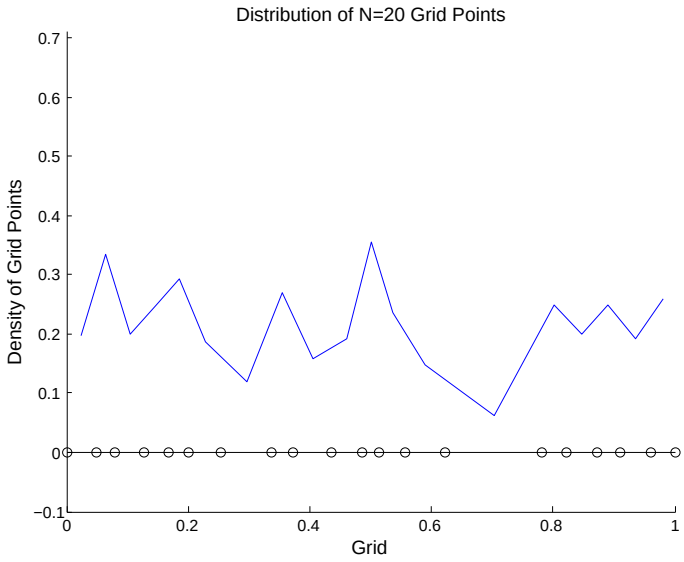


20 Grid Points

Error $1.34E-006$







3.6 $g''(x) = \frac{(\varepsilon')^2}{R(x)}$

The one-dimensional light scattering equation will be written in the Widom form as follows,

$$g''(x) = \frac{(\epsilon')^2}{R(x)}$$
$$g''(x) = \frac{1}{x} + \frac{1}{(1-x)} + \mathcal{N}$$

with solution

$$g(x) = x\ln(x) + (1-x)\ln(1-x) + \mathcal{N}^*$$

where $\mathcal{N}$ is a 'nice' function (i.e. no singularities on [0,1]), and $\mathcal{N}^*$ is the second integration of $\mathcal{N}$.

When written in the Widom form, the question becomes how to handle the singularities at $x = 0$ and $x = 1$. Since we know the solution to $g''(x) = \frac{1}{x} + \frac{1}{1-x}$ will be $g(x) = x\ln(x) + (1-x)\ln(1-x)$, we can focus on interpolating $\mathcal{N}$, and the problem reduces to solving $g''(x) = \mathcal{N}$.

4 Conclusion/Future Work

Certain factors affect the accuracy of the numeric solution for a differential equation $g''(x) = f(x)$. A general trend seen is that with a fixed number of grid points, a smaller error between $f(x)$ and $f^*(x)$ will result in a smaller error between the solution $g(x)$ and its approximation $g^*(x)$. Also, an increase in the number of interpolation grid points will tend to improve the approximation, and generally reduce the error.

Symmetric functions will, for the most part, interpolate symmetrically, although this has some variation resulting from the number of grid points used. A higher density of grid points seems to occur where the function has a high curvature, though the specific position of these points vary.

At this time, it is difficult to characterize where specifically to place grid points, although a program has been written to determine where these grid points should be placed. Eventually, it would be useful to construct a characterization of functions and optimal grid point placement.

Other future work would include extending this problem to higher dimensions. We can use the results from this one dimensional case as a basis for differential equations of the form $g_{xx}(x, y) = f(x, y)$. This allows us to focus on the two dimensional light scattering equation

$$g_{xx}(x, y) = \frac{\varepsilon_x^2 G_{yy} - 2\varepsilon_x \varepsilon_y G_{xy} + \varepsilon_y^2 G_{xx}}{G_{xx} G_{yy} - G_{xy}^2}$$

to obtain more realistic results and data.

We would like to answer the question of 'Given a differential equation $g''(x) = f(x)$, how many interpolation grid points are necessary to provide an accurate solution.' As for now, what we have is a program, but more importantly a method, which when given an input equation $g''(x) = f(x)$ and a fixed number of grid points N, will output an optimal interpolation grid and a measurement of its accuracy. We hope that this will be helpful in the future by providing recommendations on where to take data measurements.

References

[1] J. C. Lagarias, J. A. Reeds, M. H. Wright, and P. E. Wright, *Convergence Properties of the Nelder-Mead Simplex Method in Low Dimensions*, SIAM J. Optim., 9 (1998), 112–147.

[2] J. A. Nelder and R. Mead, *A simplex method for function minimization*, Computer Journal 7 (1965), 308-313.

[3] D. S. Ross, G. M. Thurston, and C. V. Lutzer, *On a partial differential equation method for determining the free energies and coexisting phase compositions of ternary mixtures from light scattering data*, The Journal of Chemical Physics 129, (2008).

[4] D. S. Ross and G. M. Thurston, *Personal correspondence*, (2008-2009).

[5] S. Singer and J. A. Nelder, *Nelder-Mead algorithm*, Scholarpedia (2008), http://www.scholarpedia.org/article/Nelder-Mead_algorithm.

A Fortran 90 Program: nminterp.f90

```
implicit none
integer j,nit,i,k,n,jj,lda,m,h

parameter(n=100, m=10)
! n finite difference grid, m interpolation grid

double precision a,b,x,y,dx,f,grid,errfact,z
double precision xx,yy,zz,solution,realf,err,r
double precision besterr,wrsterr,secerr,fexp,fcont
double precision tol,werr,fingrid
double precision nmgrid,mintemp,nmvec,nmref,nmexp,
   nmcont,fref
double precision nmerr,centroid, rowtemp, alpha,beta,
   gamma,delt
double precision gridtemp,opterr,optgrid,counter
dimension a(n,n),b(n),y(n),x(n),grid(m),errfact(m),z(n)
   ,fingrid(m)
dimension xx(n),yy(n),zz(n),r(m-2),optgrid(m)
dimension nmgrid(m,m+1),nmerr(m+1)
dimension centroid(m),nmvec(m),rowtemp(m),nmref(m),
   nmexp(m),nmcont(m)
common grid,errfact,tol,werr,nmgrid
common xx,yy,zz

open (unit=52,file="test.data",status="unknown")
open (unit=54,file="grid.data",status="unknown")
open (unit=55,file="interp.data",status="unknown")
open (unit=57,file="error.txt",status="unknown")
open (unit=58,file="grid.txt",status="unknown")


besterr=1
counter=0
write(6,*)''
```

```
write(6,*)'Calculating...'
write(6,*)''
write(6,*)''

grid(1)=0.0d00
grid(m)=1.0d00

! Setting up the grids

do j=1,m+1
   nmgrid(1,j)=0
   nmgrid(m,j)=1
enddo

do j=1,m+1
   do k=2,m-1
      nmgrid(k,j)=rand()
   enddo
enddo

! Ordering Grid points

do i=1,m
   do j=1,m+1
      do k=2,m-1
         if(nmgrid(k,j).gt.nmgrid(k+1,j)) then
            gridtemp=nmgrid(k,j)
            nmgrid(k,j)=nmgrid(k+1,j)
            nmgrid(k+1,j)=gridtemp
         endif
      enddo
   enddo
enddo


111    continue

call fcn(nmgrid(1:m,1),err)
```

```
opterr=err

counter=counter+1

if (counter == 10000) then
  goto 999
endif

! Ordering the grids

do k=1,m+1
  do j=1,m
    nmvec(j)=nmgrid(j,k)
  enddo
  call fcn(nmvec,err)
  nmerr(k)=err
enddo

do k=1,m+1
  do j=1,m
    if (nmerr(j).gt.nmerr(j+1)) then
      mintemp=nmerr(j)
      nmerr(j)=nmerr(j+1)
      nmerr(j+1)=mintemp
      do h=1,m
        rowtemp(h)=nmgrid(h,j)
        nmgrid(h,j)=nmgrid(h,j+1)
        nmgrid(h,j+1)=rowtemp(h)
      enddo
    endif
  enddo
enddo

! Calculate the centroid

do j=1,m
  centroid(j)=0
enddo
```

```
do j=1,m
  do k=1,m
    centroid(j)=centroid(j)+nmgrid(j,k)
  enddo
enddo

do j=1,m
  centroid(j)=(centroid(j))/m
enddo


! Transformations

alpha=1
beta=.5
gamma=2
delt=.5

! Calculate Reflection Point

do j=1,m
  nmref(j)=centroid(j)+alpha*(centroid(j)-nmgrid(j,m+1)
    )
enddo


call fcn(nmref,err) ! Reflection error
fref=err

call fcn(nmgrid(1:m,1),err) ! Error of best grid
besterr=err

call fcn(nmgrid(1:m,m+1),err) ! Error of the worst grid
wrsterr=err

call fcn(nmgrid(1:m,m),err) ! Error of second worst
  grid
```

```
secerr=err

if ((besterr.le.fref).and.(fref.lt.secerr)) then
  nmgrid(1:m,m+1)=nmref(1:m)
  goto 111
else if (fref.lt.besterr) then
  goto 777
else
  goto 7777
endif

777   continue

! Calculate the Expansion Point

do j=1,m
  nmexp(j)=centroid(j)+gamma*(nmref(j)-centroid(j))
enddo

call fcn(nmexp,err)
fexp=err

if (fexp.lt.fref) then
  nmgrid(1:m,m+1)=nmexp(1:m)
  goto 111
else
  nmgrid(1:m,m+1)=nmref(1:m)
  goto 111
endif

7777    continue

! Calculate the Contraction Point

if ((secerr.le.fref).and.(fref.lt.wrsterr)) then
  do j=1,m
    nmcont(j)=centroid(j)+beta*(nmref(j)-centroid(j))
  enddo
```

```
   call fcn(nmcont,err)
   fcont=err
   if (fcont.le.fref) then
     nmgrid(1:m,m+1)=nmcont(1:m)
     goto 111
   else
     goto 333
   endif
 else if (fref.ge.wrsterr) then
   do j=1,m
     nmcont(j)=centroid(j)+beta*(nmgrid(j,m+1)-centroid(
         j))
   enddo
   call fcn(nmcont,err)
   fcont=err
   if (fcont.lt.wrsterr) then
     nmgrid(1:m,m+1)=nmcont(1:m)
     goto 111
   else
     goto 333
   endif
 endif

 333    continue

 ! Shrink the Simplex

 do j=2,m+1
   do k=2,m-1
     nmgrid(k,j)=nmgrid(k,1)+delt*(nmgrid(k,j)-nmgrid(k
         ,1))
   enddo
 enddo

 goto 111

 999    continue
```

```
optgrid(1:m)=nmgrid(1:m,1)

call fcn(optgrid,err)
opterr=err

write(6,*)''
write(6,*) 'The optimal grid is:'
write(6,*) ''
write(6,*) optgrid
write(6,*)''
write(6,*) 'With minimum error:',opterr
write(6,*)''
write(6,*)''

call fcn(optgrid,err)

write (52,10) (xx(j),j=1,n), opterr
write (52,10) (yy(j),j=1,n)
write (52,10) (zz(j),j=1,n)
write (52,10) (optgrid(j), j=1,m)
write (52,10) (0.0*grid(j), j=1,m)
write (54,10) (optgrid(j), j=1,m)
write (54,10) (0.0*grid(j), j=1,m)
write (58,10) (optgrid(j), j=1,m)
write (55,10) (yy(j),j=1,n)
write (55,10) (zz(j),j=1,n)
write(57,*) opterr

10      format (505(' ',f13.8))
end

! ***Finite difference***

subroutine fcn(griddd,err)
implicit none
integer j,nit,i,k,n,lda,m
parameter(n=100, m=10)
```

```
double precision a,b,x,y,dx,f,grid,griddd,tol,werr
double precision xx,yy,zz
double precision errfact,z, solution, realf, err,nmgrid
dimension a(n,n),b(n),y(n),x(n),grid(m),griddd(m),
    nmgrid(m,m+1)
dimension xx(n),yy(n),zz(n)
dimension errfact(m),z(n)
common grid,errfact,tol,werr,nmgrid
common xx,yy,zz

grid(1)=0.0
grid(m)=1.0
do i=2,m-1
   grid(i)=griddd(i)
enddo
lda=n
dx=1.0d00/dfloat(n+1)
do i=1,n
   do j=1,n
      a(i,j)=0.0d00
   enddo
   x(i)=i*dx
   b(i)=dx*dx*f(x(i))
   z(i)=solution(x(i))
enddo
a(1,1)=-2.0d00
a(n,n)=-2.0d00
a(1,2)=1.0d00
a(n,n-1)=1.0d00
do j=2,n-1
   a(j,j)=-2.0d00
   a(j,j+1)=1.0d00
   a(j,j-1)=1.0d00
enddo

! Solving the approximate solution

call solver(n,a,lda,b,y)
```

```
! ***Calculating error between z and y***

err=0.0
do j=1,n
  err=err+(y(j)-z(j))**2
enddo
err=sqrt(err*dx)

do j=1,n
  xx(j)=x(j)
  yy(j)=y(j)  !Approximate solution
  zz(j)=z(j)  !Actual solution
enddo

10      format (505('␣',f13.8))
end


! ***Interpolation of function f***

double precision function f(x)

implicit none
integer n,m,j ,i
parameter(n=100, m=10)
double precision x,z,grid, realf,errfact,tol,werr,
   nmgrid
dimension grid(m),errfact(m),nmgrid(m,m+1)
common grid,errfact,tol,werr,nmgrid

if (x.le.grid(1)) then
  f=realf(x)
else
  do j=2,m
    if (x.le.grid(j)) then
      f=(realf(grid(j-1))*(grid(j)-x)+realf(grid(j))*(x
        -grid(j-1)))/(grid(j)-grid(j-1))
```

```
          go to 12
      endif
    enddo
endif

12     continue

return
end

! ***Actual function f***

double precision function realf(x)

implicit none
integer n,m
parameter(n=100, m=10)
double precision x

realf=(x-(x**2))

return
end

! ***Actual solution g***

double precision function solution(x)

implicit none
integer n,m
parameter(n=100, m=10)
double precision x,y

solution=((x**3)/6)-((x**4)/12)-(x/12)

return
end
```

```
! ***Solver for the system of equations***

Subroutine solver(n,a,lda,b,y)

integer n,lda,j
double precision a,y,b,r
dimension a(n,n),b(n),y(n)

do j=2,n
   r=a(j,j-1)/a(j-1,j-1)
   a(j,j)=a(j,j)-r*a(j-1,j)
   b(j)=b(j)-r*b(j-1)
enddo

y(n)=b(n)/a(n,n)

do j=n-1,1,-1
   y(j)=(b(j)-a(j,j+1)*y(j+1))/a(j,j)
enddo

End Subroutine solver
```

B MATLAB Plotting Code

```
M=load('interp.data'); % loading functions and
    solutions
G=load('grid.data'); % loading the grid

Mtemp=zeros(3,length(M)+2);


 for k=1:size(M)
      for j=1:length(M)
           Mtemp(k+1,j+1)=M(k,j);
      end
 end

 for h=1:length(Mtemp)-1
     Mtemp(1,h+1)=Mtemp(1,h)+(1/101);
 end

M=Mtemp;
f=M(4,:);

g1=G(1,:);
g2=G(2,:);

diffgrid=M(1,:); % difference grid
y=M(2,:); % approx
z=M(3,:); % actual

% Only used for polynomial graphs

poly=[-1 1 0];
actsol=polyval(poly,diffgrid);


% Solution Plot
figure
```

```
hold on
xlim([0,1])
plot(diffgrid,y,'r'); %aprox
plot(diffgrid,z,'b'); %actual
plot(g1,g2,'-k',g1,g2,'ok');
title(['Actual and Approximate Solutions to g\prime\
    prime(x) = x-x^2'],'FontSize',12)
xlabel('x','FontSize',12);
ylabel('g and g Approximate','FontSize',12);


n=20; % number of grid points

% Setting up the distribution plot

lennnn=length(g1)-1;

dist=zeros(1,lennnn);
invdist=zeros(1,lennnn);
mid=zeros(1,lennnn);

for i=1:n-1
     dist(i)=((G(1,i+1)-G(1,i)));
     invdist(i)=(1/((G(1,i+1)-G(1,i))));
     mid(i)=(G(1,i+1)+G(1,i))/2;
end

normmm=norm(invdist);
normmmm=norm(dist);

for i=1:n-1
     invdist(i)=(invdist(i)/normmm);
     dist(i)=(dist(i)/normmmm);
end

maxii=max(invdist);

% Grid point distribution plot
```

```
figure
hold on
ylim([-0.1,2*maxii])
plot(mid,invdist,'b');
plot(g1,g2,'-k',g1,g2,'ok');
title(['Distribution of N=20 Grid Points'],'FontSize'
    ,12)
xlabel('Grid','FontSize',12)
ylabel('Density of Grid Points','FontSize',12)

% Function plot
figure
hold on
xlim([0,1])
plot(diffgrid,f,'r')
plot(diffgrid,actsol,'b')
plot(g1,g2,'-k',g1,g2,'ok');
title(['g\prime\prime(x) = x-x^2 and Interpolation'],'
    FontSize',12)
xlabel('x','FontSize',12);
ylabel('f and f Approximate','FontSize',12);
```