Weekly Status Report

Development of a 1D Adaptive Wavelet Collocation Code
Brandon Gusto

Week Beginning 8/28/2017

1 Summary of method

The adaptive wavelet collocation method for differential equations has a number of merits which may make it a viable alternative to traditional finite element, finite volume, or high order finite difference methods for certain problems. Some important advantages include the following:

- the multiresolution properties of wavelets allow for natural grid adaptation
- the existence of a fast $\mathcal{O}(\mathcal{N})$ scheme for computation of wavelet coefficients and spatial derivatives
- the ability of second-generation wavelets to adapt to complex geometries

1.1 Dyadic Grid

The method makes use of a dyadic grid. Each grid level is defined by

$$\mathcal{G}^j = \{x_k^j \in \Omega : k \in \mathcal{K}^j\}, j \in \mathcal{Z}, \tag{1}$$

where $\mathcal{K}^j$ is the integer set representing the spatial locations in the grid at level j. The grids are nested, implying that $\mathcal{G}^j \subset \mathcal{G}^{j+1}$. In other words, the points at level x^j are a perfect subset of the points at level x^{j+1}. This can also be demonstrated by the relation that $x_k^j = x_{2k}^{j+1}$.

1.2 Interpolating Subdivision

The interpolating subdivision scheme is central to the second-generation wavelet collocation approach. The scheme is used to approximate values at odd points x_{2k+1}^{j+1} by constructing interpolating polynomials of order $2N-1$ with the points at grid level j. Lagrange polynomials are used, and the method can be used with a uniform grid or with nonuniform points such as Chebychev points. The interpolating scheme is

$$f^j(x_{2k+1}^{j+1}) = \sum_{l=-N+1}^{N} w_{k,l}^j f(x_{k+l}^j), \tag{2}$$

where the coefficients $w_{k,l}^j$ are the lagrange polynomials $L_{2N,k+l}(x)$ evaluated at $x = x_{2k+1}^{j+1}$. The lagrange polynomial in this notation is given by

$$L_{2N,k+l}(x) = \prod_{\substack{i=k-(N+1) \\ i \neq k+l}}^{k+2N} \frac{x - x_i}{x_{k+l} - x_i} \tag{3}$$

For a uniform grid, the coefficients should be constant irrespective of the grid level j.

1.3 Fast Wavelet Transform

The existence of a fast wavelet transform is one of the more attractive qualities of the method. The transform makes use of the interpolating subdivision algorithm for the calculation of the scaling and detail wavelet coefficients. The forward wavelet transform is given by

$$\begin{aligned} d_k^j &= \frac{1}{2}\left(c_{2k+1}^{j+1} - \sum_l w_{k,l}^j c_{2k+2l}^{j+1}\right), \\ c_k^j &= c_{2k}^{j+1}, \end{aligned} \tag{4}$$

and the inverse transform is given by

$$\begin{aligned} c_{2k+1}^{j+1} &= 2d_k^j + \sum_l w_{k,l}^j c_{k+l}^j, \\ c_{2k}^{j+1} &= c_k^j. \end{aligned} \tag{5}$$

Note that this second-generation wavelet transform does not include the lifting scheme. When using the lifting scheme, polynomials are also constructed from odd to even points.

1.4 Wavelet Construction

The construction of second-generation wavelets makes use of the dyadic grid. The interpolating subdivision scheme is used to interpolate functional values defined at points on level j, to odd points (i.e. x_{2k+1}^{j+1}) at the next higher level of resolution. This scheme is used to construct the scaling and detail wavelet functions. Examples of the scaling and detail functions are shown in Figure 1 and Figure 2 respectively. The computation of the wavelet functions is done using the inverse wavelet transform. To obtain the scaling function $\phi_m^j(x)$, from (5) set $c_k^j = \delta_{k,m}, \forall k \in \mathcal{K}^j$, where $\delta_{k,m}$ is the Kronecker delta function defined by

$$\delta_{k,m} = \begin{cases} 1 & k = m \\ 0 & k \neq m. \end{cases}$$

Then let all $d_l^{j'} = 0, \forall l \in \mathcal{L}^{j'}$ and perform the inverse transform up to an arbitrarily high level of resolution J. The wavelet ψ_l^j is computed by setting $d_m^{j'} = \delta_{j',j}\delta_{l,m}, \forall l \in \mathcal{L}^j, \forall j \geq j$, and also $c_k^j, \forall k \in \mathcal{K}^j$. Then perform the inverse wavelet transform up to an arbitrarily high level of resolution J.

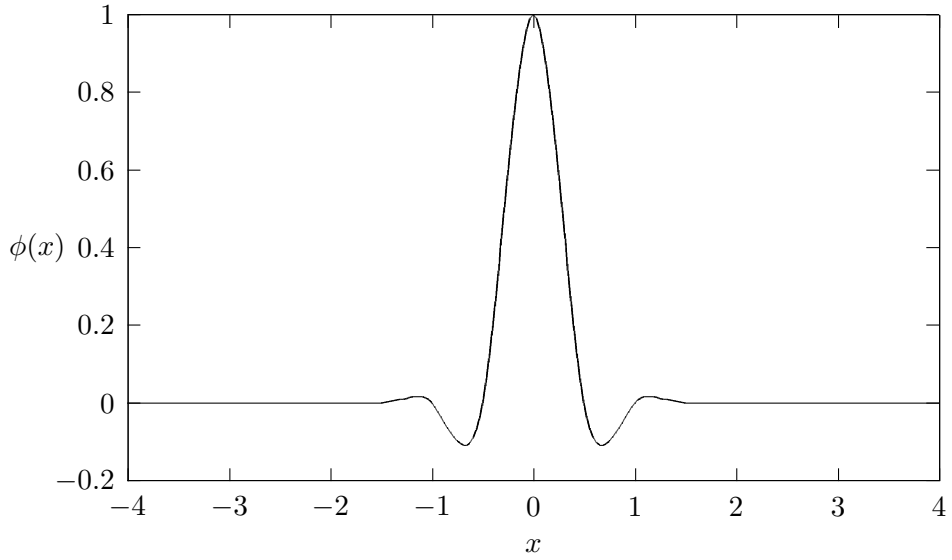

Figure 1: An example of a scaling wavelet, $\phi(x)$, for $N = 3$.

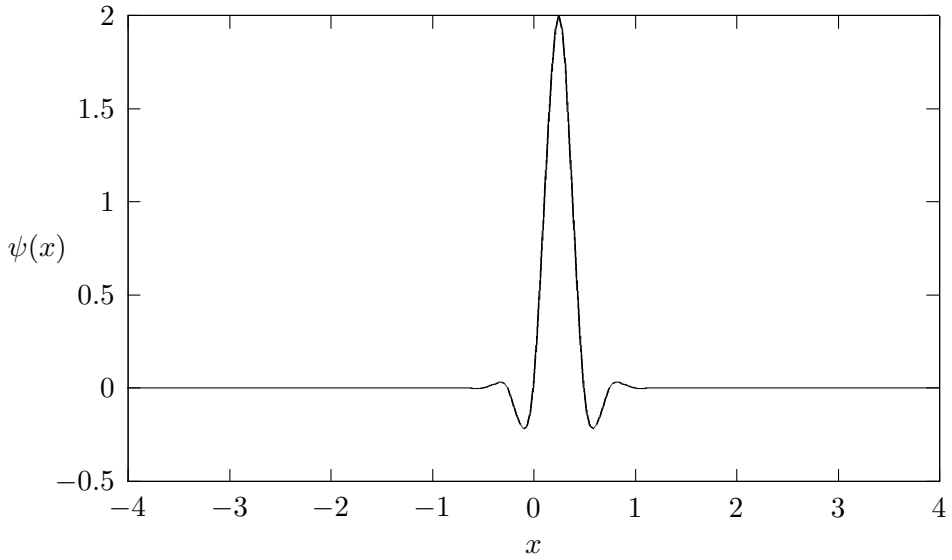

Figure 2: An example of a daughter wavelet, $\psi(x)$, for $N = 3$.

1.5 Approximation of Functions

The approximation of a function $f(x)$ is done by setting the scaling coefficients at the arbitrary maximum level of resolution J to the function itself. Once the function is sampled this way on $\mathcal{G}^J$, the forward wavelet transform is performed down to the coarsest level of resolution. The function is then represented by

$$f^J(x) = \sum_{k \in \mathcal{K}^0} c_k^0 \phi_k^0(x) + \sum_{j=0}^{J-1} \sum_{l \in \mathcal{L}^j} d_l^j \psi_l^j(x). \tag{6}$$

Often, a large number of wavelet coefficients can be discarded, and the approximation (6) still is adequate. Define some threshold ϵ for the coefficients, then keep only those coefficients which satisfy $|d_l^j| \geq \epsilon$. The approximation (6) becomes

$$f_{\geq}^J(x) = \sum_{k \in \mathcal{K}^0} c_k^0 \phi_k^0(x) + \sum_{j=0}^{J-1} \sum_{\substack{l \in \mathcal{L}^j \\ |d_l^j| \geq \epsilon}} d_l^j \psi_l^j(x). \tag{7}$$

In the figures below are approximations to the function $f(x) = \cos(80\pi x)\exp(-64x^2)$ with an increasing threshold ϵ.

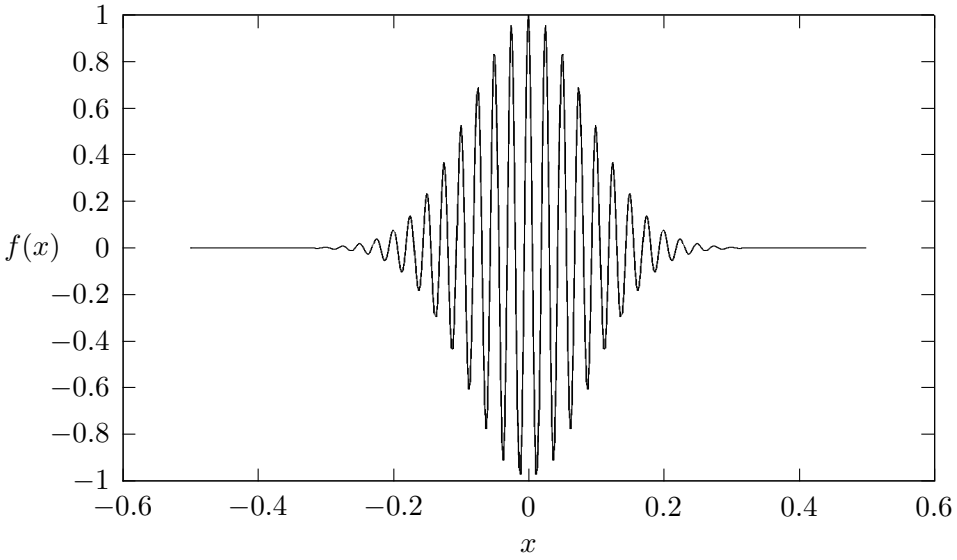

Figure 3: Approximation of $f(x) = \cos(80\pi x)\exp(-64x^2)$, with $N = 3$ and 256 points on $\mathcal{G}^J$. All coefficients are kept.

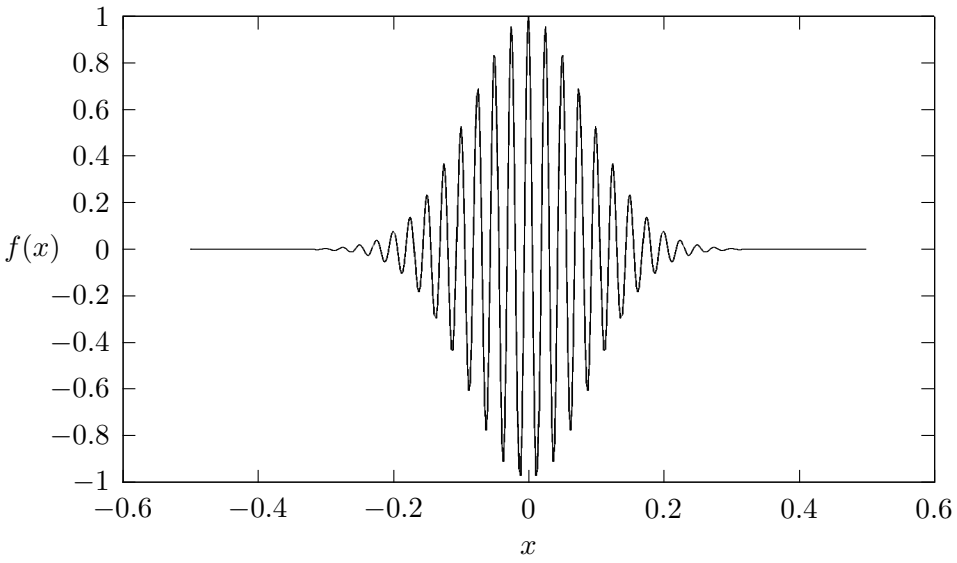

Figure 4: Approximation of $f(x)$ with $\epsilon = 10^{-5}$.

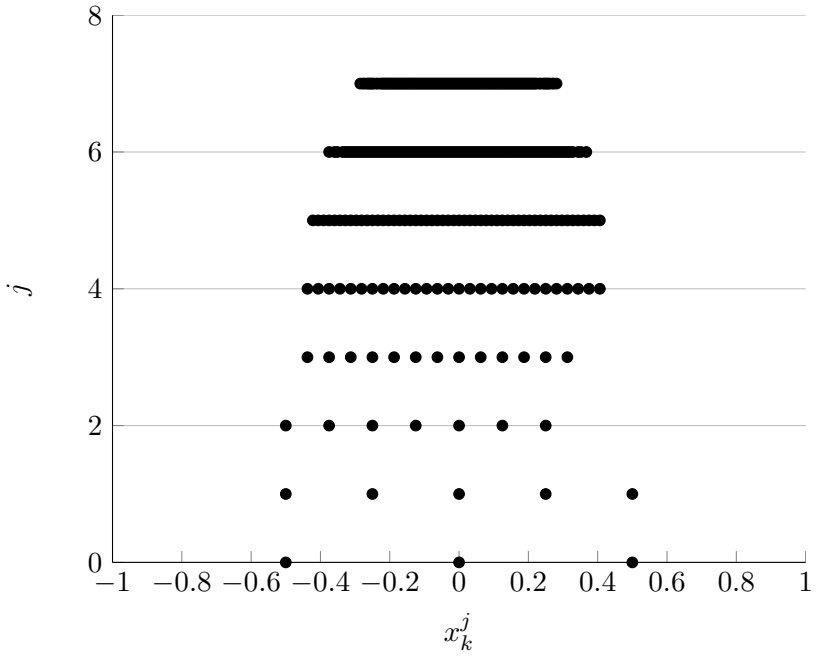

Figure 5: Coefficients kept with $\epsilon = 10^{-5}$.

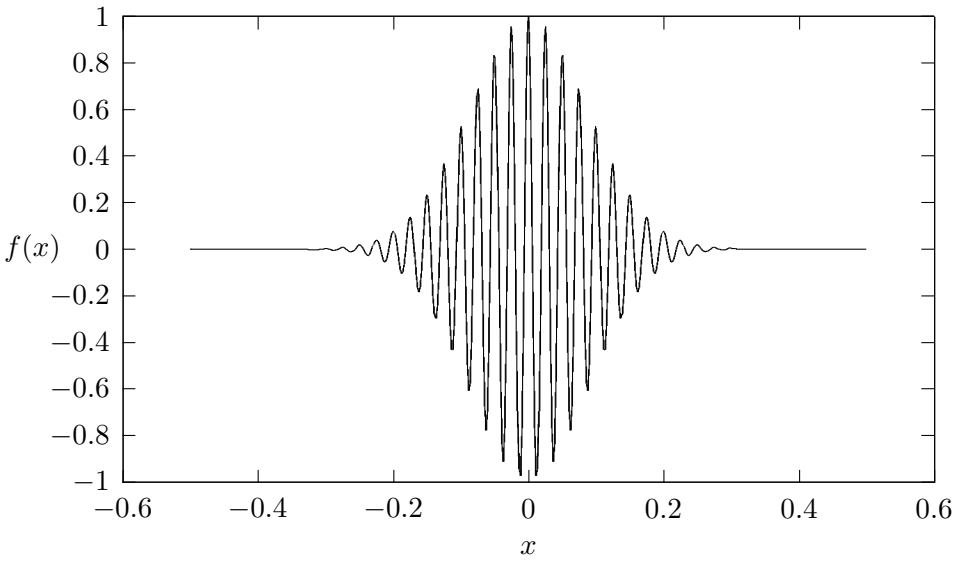

Figure 6: Approximation of $f(x)$ with $\epsilon = 10^{-3}$.

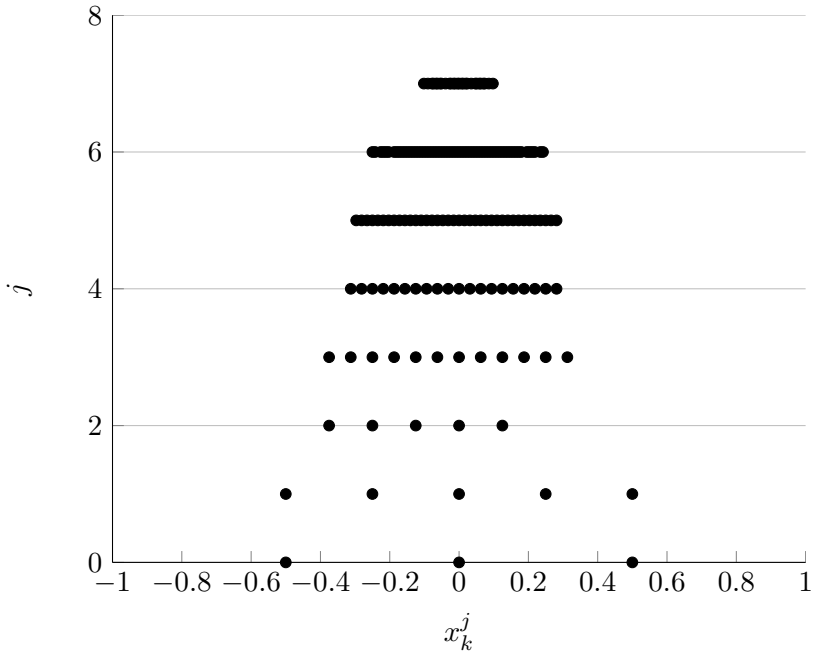

Figure 7: Coefficients kept with $\epsilon = 10^{-3}$.

2 To Be Completed Soon

1. Implement code for calculating spatial derivatives. When a particular point on $\mathcal{G}^{j+1}$ is absent of its detail coefficient (due to the threshold, or $j+1$ is the maximum level of refinement), then the approximation from the interpolating polynomial constructed on $\mathcal{G}^{j}$ is good enough. Thus differentiating the local polynomial is a good enough approximation to the derivative of the function at that location.

2. The issue of the detail coefficient at the last point in the grid must be solved. As can be seen in Figure 5 and Figure 7, the detail coefficients are not being calculated correctly at the very last index. This would depend on interpolating at a ghost point possibly, so I am determining the best way to deal with the issue.

3. The additional 'lifting' step requires adjustment of 3 lines of code in the program. One simply also constructs interpolating polynomials from the odd points to the even points. I wanted to get the code working properly before moving on to this necessary step.