The FEMhub Project and Classroom Teaching of Numerical Methods

Pavel Solin (solin@unr.edu) – *University of Nevada, Reno,, USA*
Ondrej Certik (ondrej@certik.cz) – *University of Nevada, Reno,, USA*
Sameer Regmi (sregmi@unr.edu) – *University of Nevada, Reno,, USA*

We introduce briefly the open source project FEMhub and focus on describing how it can be used for live demonstrations of elementary numerical methods in daily classroom teaching.

The FEMhub Project

FEMhub [femhub] is an open source distribution of finite element (FEM) codes with unified Python interface, developed by the *hp*-FEM group at the University of Nevada, Reno [hpfemorg]. The aim of FEMhub is to establish common standards in the development of open source FEM codes, allow for accuracy and performance comparisons of different codes, and provide a common platform for collaboration and exchange of modules.

Currently, FEMhub contains the open source codes FiPy, Hermes [hermes], Phaml and SfePy as FEM engines, tools to ease visualisation (matplotlib [mpl], mayavi [mayavi], pyglet [pgl]), standard Python libraries Scipy [scipy], Numpy [numpy] and Sympy, and a web notebook which is based on the Sage notebook.

Interactive Web Notebook

The goal of the FEMhub web notebook [femhub-nb] is to make all FEM codes in FEMhub available remotely through any web browser. Inside the web notebook, one will be able to define geometry, generate a mesh, specify boundary and initial conditions, define arbitrary partial differential equations (PDE) to be solved, package the components and send them for processing to a remote high-performance computing facility (currently UNR Research Grid), and visualize the results once they are received.

Teaching Numerical Methods

We have found students' understanding of the fundamentals of numerical methods to be an obstacle to learning *hp*-FEM algorithms efficiently. As a result, we decided to use the web notebook to implement a series of elementary numerical methods that the students can work with interactively and thus gain a much better understanding of the material. The notebook does not employ the CPU of the machine where it is executed, and therefore one can use it to compute on desktop PCs, laptops, netbooks and even iphones. In particular, one can use it in every classroom that has Internet access.

The response of the students was very positive, therefore we started to add new worksheets systematically and to utilize the notebook in the classroom regularly. We found that by running the methods in real time, we can get much more across about their strengths, weaknesses and typical behavior than ever before. Last but not least, the students stopped grumbling about programming homework assignments.

Through this paper, we would like to share our positive experience with anyone who teaches elementary numerical methods. All worksheets described below are freely available at http://nb.femhub.org. Right now (as of November 2009) you still need to create an account to access them, but we are working currently on eliminating this and making the notebook even more open.

Taylor Polynomial

The Taylor polynomial $T(x)$ is an approximation to a function $f(x)$ in the vicinity of a given point a, based on the knowledge of the function value $f(a)$, first derivative $f'(a)$, second derivative $f''(a)$, etc. In the web notebook, one has two ways of defining $T(x)$: Via the point a, list *[f(a), f'(a), f''(a), ...]*, and the endpoints:
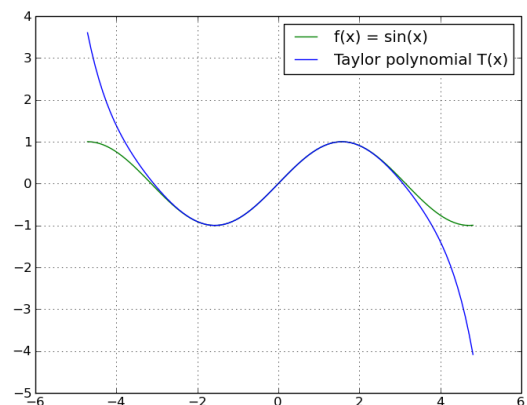
    taylor_1(0, [0, 1, 0, -1, 0, 1, 0, -1], -3*pi/2, 3*pi/2)
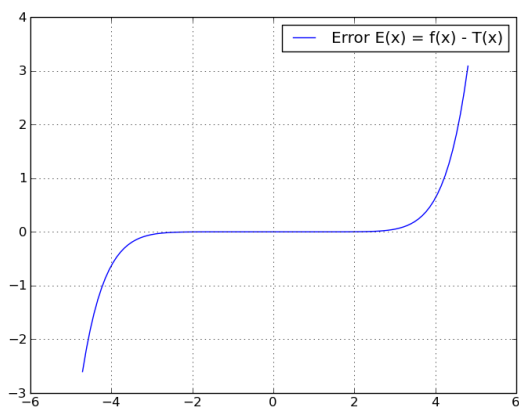
or by entering the point a, the function $f(x)$, x for independent variable, degree n, and the endpoints:

    taylor_2(0, sin(x), x, 7, -3*pi/2, 3*pi/2).

Both these commands generate the same $T(x)$ but the latter also plots the error:

    f(x) = sin(x)
   T(x) = x - x**3/6 + x**5/120 - x**7/5040

Using the notebook, one can demonstrate obvious facts such as that with increasing n the polynomial $T(x)$ gets closer to $f(x)$ in the vicinity of the point a, but also that the textbook recipe may overestimate the error $|f(x) - T(x)|$ substantially, that $T(x)$ usually diverges from $f(x)$ quickly once $|x-a|$ exceeds some threshold, that $T(x)$ has a special form at $a=0$ if $f(x)$ is even or odd, etc. See the Taylor Polynomial worksheet at [femhub-nb] for more details.

Rootfinding Techniques

The three most-widely used methods (interval bisection, Newton's method, and fixed-point iteration) are called as one would expect:

```
bisection(1/(1+x*x) - x, x, 0, 1, 1e-8)
newton(1/(1+x**2) - x, x, 1, 1e-8)
fixed_point(1/(1+x**2), x, 1, 1e-8)
```

For the bisection method, we like to illustrate the textbook formula for the number of steps needed to obtain the root with a given accuracy. For the Newton's method we show how extremely fast it usually is compared to the other two techniques but also that it can fail miserably when the initial guess is far from the root. For the fixed-point iteration we like to show how slow it typically is compared to the other two methods, but also that it may work in situations where the Newton's does not. And of course, that it can fail if the derivative of the underlying function exceeds the interval *(-1,1)*.
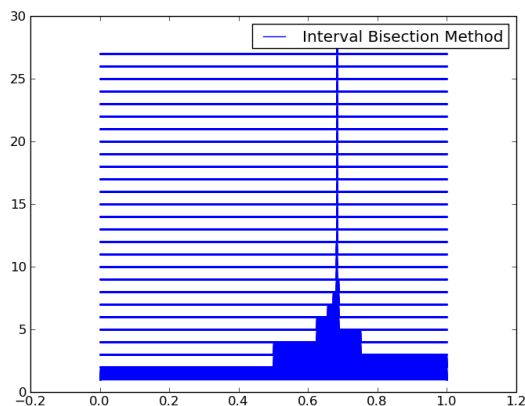

The previous image illustrates the history of interval subdivisions produced by the bisection method.
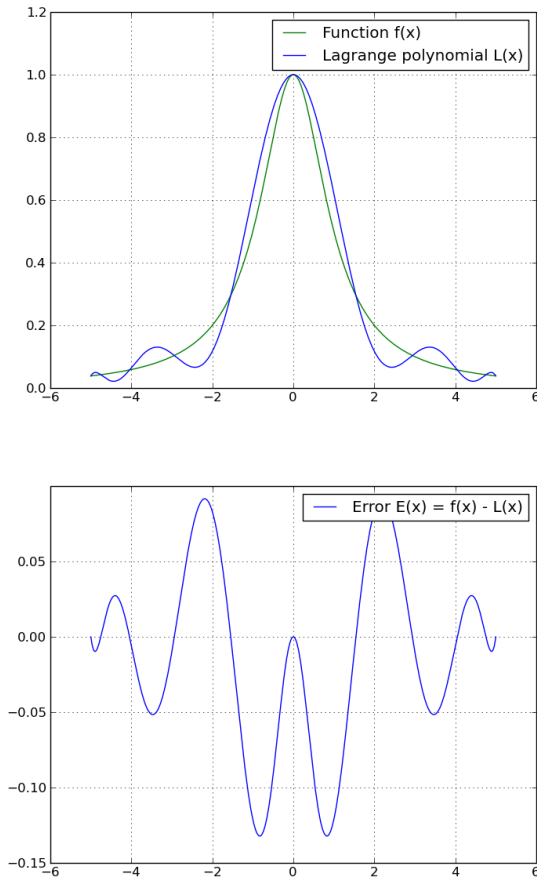
Lagrange Interpolation Polynomial

The Lagrange interpolation polynomial $L(x)$ is a single polynomial of degree n passing through $n+1$ given points of the form [x, y] with distinct x-coordinates. The worksheet offers four different problem types:

1. Given is an arbitrary array of x-coordinates and an arbitrary array of y-coordinates.

2. Given is an arbitrary array of x-coordinates and a function $f(x)$ for y-coordinates.

3. Given is an array of equally-spaced x-coordinates and a function $f(x)$ for y-coordinates.

4. Given are Chebyshev points for x-coordinates and a function $f(x)$ for y-coordinates.

We like to use option #1 to show the elementary interpolation functions that are equal to one at one of the points and zero at the remaining ones. Option #2 is useful for showing the difference between the Lagrange and Taylor polynomials. Options #3 and #4 can be used to demonstrate the notoriously bad performance of equally-spaced interpolation points and that one should use the Chebyshev points instead. The following two figures illustrate interpolation of the famous function $1/(1+x**2)$ in the interval (-5, 5) on 11 equidistant points and the huge error that one ends up with:
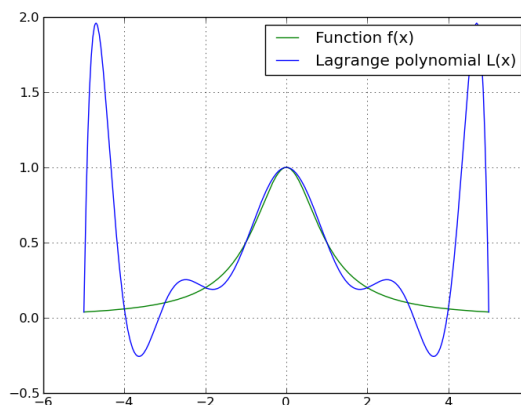
The following pair of figures shows the same situation, only the equidistant points are replaced with Chebyshev points. The reader can see that the error $E(x) = f(x) - L(x)$ drops about 13 times in magnitude:


In option #4, one can demonstrate nicely the optimality of the Chebyshev points by moving slightly some of them to the left or right (in the part of the code where they are actually defined) and observing that the interpolation error always goes up. See the Lagrange Polynomial worksheet at [femhub-nb] for more details.
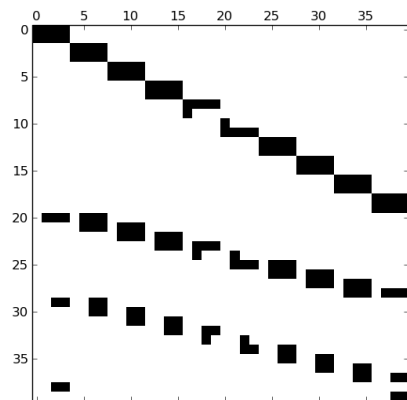
Cubic Splines

Interpolation via cubic splines is more popular than the Lagrange interpolation because the results do not contain wiggles and look much more natural. The basic setting is the same as in Lagrange interpolation - one has $n+1$ points of the form $[x,y]$ with distinct x-coordinates. These points divide the interval into n subintervals. In each of them, we seek a cubic polynomial via four unknown coefficients. This means that we have $4n$ unknowns. The corresponding $4n$ equations are obtained by requiring the splines to match the end point values in every subinterval ($2n$ equations), requiring the first derivatives from the left and right to be the same at all interior points ($n-1$ equations), and the same for second derivatives (another $n-1$ equations). At this point, two conditions are left to be specified and there is some freedom in them. For example, one can require the second derivatives

to vanish at interval endpoints, which results into the so-called *natural cubic spline*. But one can also set the first derivatives (slopes) at interval endpoints, etc. The following pair of images illustrates the interpolation of the function $1/(1+x*x)$ in the interval $(-5, 5)$ on 11 equidistant points as above, but now with cubic splines. Compared to the Chebyshev interpolant, the error drops 6 times in magnitude.
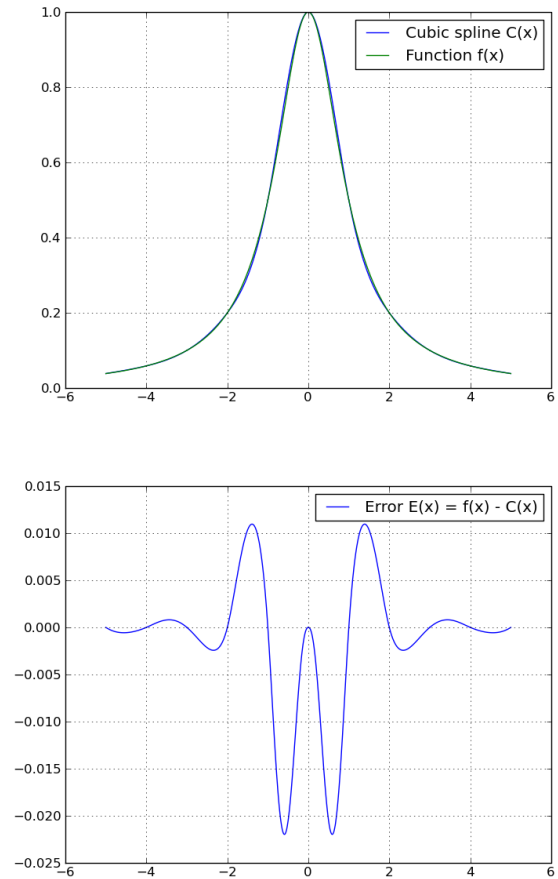
The following figure shows the sparsity structure of the $4n$ times $4n$ matrix (zeros in white, nonzeros in black). We like to highlight the underlying matrix problems because in our experience the students usually do not know that matrices can be used outside of a linear algebra course.

See the Cubic Splines worksheet at [femhub-nb] for more details.

Least-Squares Approximation

The least-squares approximation technique is very different from interpolation - it finds in a given set of polynomials an element that is *closest to the approximated function f(x)*. (There is nothing about matching the function values of $f(x)$ exactly.) Typically, the set of polynomials is chosen to be the Legendre polynomials because of their orthogonality, and the distance between is measured in the so-called *L2-norm*. The following commands are used to define a function $f(x)$ and calculate its least-squares polynomial approximation of degree n in an interval (a,b):

```
# Define function f(x)
def f(x): return -sin(x)

# Calculate and plot in interval (a, b) the
# least-squares polynomial approximation P(x) of f(x)
least_squares(-pi, pi, f, x, 3)
```

The output for these parameters looks as follows:
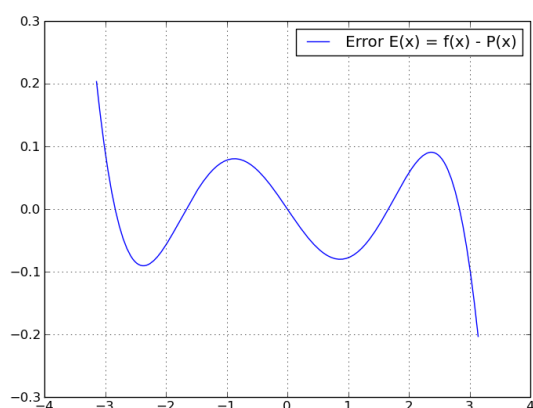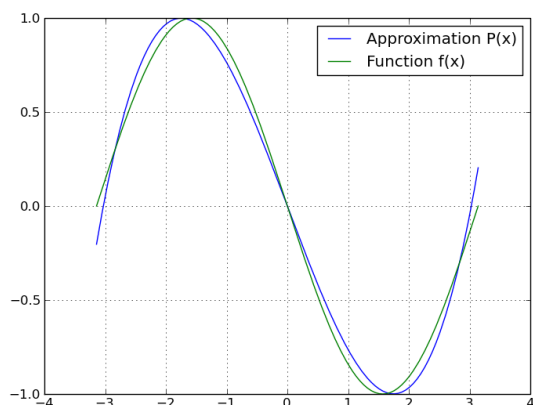
This worksheet also plots the underlying basis functions (Legendre polynomials). One can use elementary integration functions to show the students that indeed the Legendre polynomials are orthogonal in the $L2$-product. We also like to use this opportunity to explain the importance of numerical quadrature by projecting a complicated function that cannot be integrated analytically. For more details see the Least Squares worksheet at [femhub-nb].

Fourier Expansion

This is an excellent opportunity to show the students that the Fourier expansion is nothing else than the least-squares approximation. One just replaces the Legendre polynomials with the functions *1, cos(x), sin(x), cos(2x), sin(2x), ...*, and considers the periodicity interval *(-pi, pi)* instead of a general interval *(a, b)*. Otherwise everything is the same. It is useful to demonstrate to the students that the Fourier basis above indeed is orthogonal in the $L2$-product. The following figure shows the approximation of a piecewise-constant discontinuous signal. The worksheet also plots the error as usual, not shown here for space limitations.
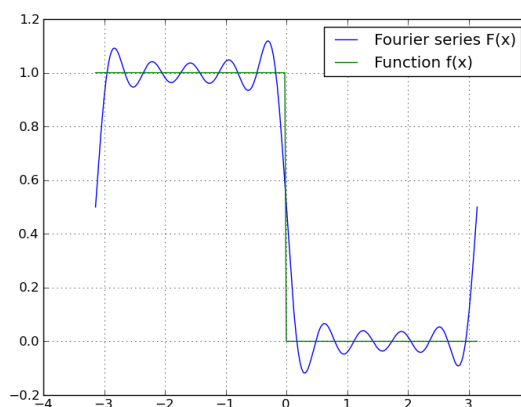
See the Fourier Expansion worksheet at [femhub-nb] for more details.

References

[femhub]   http://femhub.org/.
[femhub-nb] http://nb.femhub.org/.
[hermes]   http://hpfem.org/main/hermes.php.
[hpfemorg]   http://hpfem.org/.
[mpl]   J.D. Hunter (2007). *Matplotlib: A 2D graphics environment.* Computing in Science and Engineering. 9: 90-95. http://matplotlib.sourceforge.net/.
[mayavi]   P. Ramachandran, G. Varoquaux, *Mayavi: Making 3D Data Visualization Reusable*, in Proceedings of the 7th Python in Science conference (SciPy 2008) http://code.enthought.com/projects/mayavi/.
[numpy]   T. Oliphant et al., *NumPy*, http://numpy.scipy.org/.
[pgl]   http://www.pyglet.org/.
[scipy]   E. Jones, T. Oliphant, P. Peterson, *SciPy: Open source scientific tools for Python* http://www.scipy.org/.