Decomposition in basis functions.

Moritz Wolter

June 5, 2016

1 Abstract

The application developed in this project approximates requested function values on the basis of given sampled function data. Approximations are obtained by application of Lagrange interpolation, orthogonal basis inter and extrapolation and finally by projection. These methods need numerical integration routines to work. Therefore several numerical integration schemes have been implemented in this project.

2 Used Algorithms

2.1 Lagrange

The most basic approximation method used was plain Lagrange approximation. This methods interpolates missing points of a sampled function $f(x)$ of which only the sampled values $f_1(x_1) \dots f_n(x_n)$ and the points $x_1 \dots x_n$ are known. In general the Lagrange polynomials are defined as:

$$l_i = \Pi_{j=0, j\neq i}^{n} \frac{x - x_j}{x_i - x_j}, \qquad i = 0, 1, \dots, n. \tag{1}$$

However this methods requires $O(n^2)$ operations in every point, which is not very efficient. A more efficient implementation uses the *Barycentric Formula*[1]. To use this formula lambda coefficients have to be defined:

$$\lambda_i = \frac{1}{(x_i - x_0) \dots (x_i - x_{i-1})(x_i - x_{i+1}) \dots (x_i - x_n)} \tag{2}$$

With weights defined as $\mu_i = \lambda_i/(x - x_i)$, the Lagrange polynomials can be computed from:

$$l_i = \frac{\mu_i}{\sum_{i=0}^{n} \mu_i(x)}, \qquad i = 0, 1, \dots, n. \tag{3}$$

Which allows for interpolation in O(n) steps. However when a high degree is used the interpolation polynomial becomes unstable (more on this follows in the section 3).

[1] See W.Gander, Change of Basis in polynomial interpolation

2.2 Orthoginal basis approximation

2.2.1 Monomial basis

To fix the stability problem with higher order approximations an orthogonalization method is required. Before working with Lagrange-Polynomials a monomial polynomials like $1, x, x^2, x^3, \ldots, x^M$ will be considered for simplicity. Orthogonalization is done with respect to two essential definitions:

$$\int_{-1}^{1} w(x)p_n(x)p_k(x)dx = 0, \quad k \neq n, \tag{4}$$

$$\int_{-1}^{1} w(x)p_n(x)^2dx = 1. \tag{5}$$

Which leads to this pseudocode implementation:

```
function [orthpol] = gramSchmodt(alpha,beta,polMat)
    for i=2:#cols
        %normalize the previous column.
        polMat(:,i-1) = polMat(:,i-1) ./ sqrt(integral(norDenFun,-1,1))

        %orthogonalize
        for j=1:(i-1)
                parPart = integral(numFun,-1,1)
                / integral(norDenFun,-1,1) * polMat(:,j);
                polMat(:,i) = polMat(:,i) - parPart
          end
        end
end
 %nomalize the last column
 polMat(:,end) = polMat(:,end) ./ sqrt(integral(normFun,-1,1));

contains
        %the weight function
        omega = @(x) (1 - x)^alpha * (1 + x)^beta
        %for normalizing and computing the denominator
        norDenFun = @(x) omega(x) * polyval(:,i-1)^2
        %for computing the numerator.
        numFun = @(x) omega(x) * polyval(polMat(:,i),x)
                        *polyval(orthpol(:,j),x);
```

A slow row based `matlab` implementation of this code can be found in the test code that comes with this project. The column based version described here is implented in the `fortran` code. If this function is run on monimials, that means on an identity matrix with its rows flipped upside down[2]. The basis shown in figure 1 is obtained.

[2] flipud(eye(N))

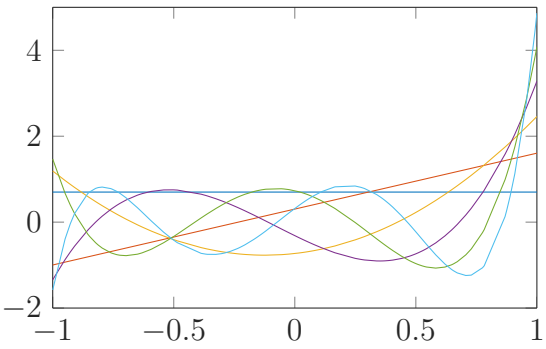

Figure 1: Monomial orthogonal basis.

2.2.2 Lagrange basis

Next we will use the function described above the listing above to orthogonalize lagrangian polynomials. A set of lagrangian polynomials my be obtained by using polynomial convolution:

```
function lagPoly = lagrange(deg)
x = linspace(-1,1,deg);
for i =1:deg
  denom = 1;
  nom = 1;
  for j =1:length(x)
    if (i ~= j)
      denom = denom * 1/(x(i) - x(j));
      nom = conv(nom, [1 -x(j)]);
    end
  end
  lagPoly(:,i) = nom .* denom;
end
```

Where the function `conv` computes the coefficients of the product of two polynomials according to:

```
function res = conv(p1,p2)
    do i = 1,length(p1)
      conMat(i,i:(i+length(p2))) = p1(i) * p2
    end do

    do i = 1,(length(p1)+length(p2)-1
      res(i) = sum(conMat(:,i))
    end do
end function
```

If the obtained polynomials are fed in to the Gram-Schmidt algorithm and plotted afterwards figure 2 is obtained.

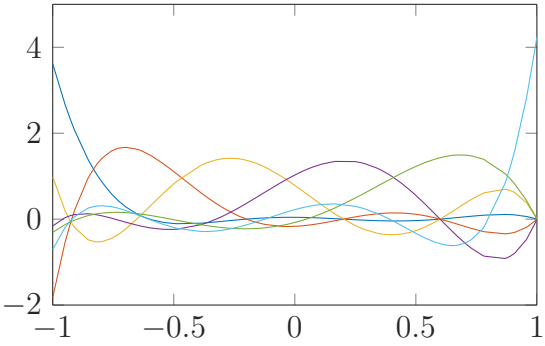

Figure 2: Lagrangian orthogonal basis.

2.3 Function-Representation

Next the representation of the sampled function has to be estimated. This may be done by solving a least squares problem of by projection. When following the least squares approach the basis polynomials are first evaluated in the known points. The resulting matrix $\mathbf{P}$ and the function values are then used to solve:

$$\mathbf{c} = \mathbf{P}^{\dagger} \cdot \mathbf{f}^{T} \text{ or } min \|\mathbf{Pc} - \mathbf{f}\|_2. \tag{6}$$

Which my be done by using lapack's `sgelss` routine.

2.4 Sorting

Following the well established principle to never trust ones input, it is assumed that values, which are read in are not necessarily sorted. In most tables, databases or other online data sources that this code could be used with the output values will be sorted. Therefore Bubble-Sort was chosen as the sorting algorithm due to its unique ability to detect sorted inputs after n operations. Quicksort for example works $nlog(n)$ operations on sorted data.

2.5 Software development methods

During the development of this project written code has been subject to continuous testing. A set of self evaluating bash scripts has been used for testing purposes. These scripts test important parts of the code[3]. Often the results obtained by fortran code are compared to the return values of trusted mathlab/octave code. Git was used to handle version control, throughout the first part a reversion was not necessary. But the private on-line backup would have been useful in case of hardware failure, with resulting data loss.

3 Results

Approximation results for orthogonal inter- and exrapolation with monimial and lagrange-basis are shown in figure 3. Clearly the orthogonal polynomials don't suffer as much from

[3]Please run `./tests/testAll_run.sh`

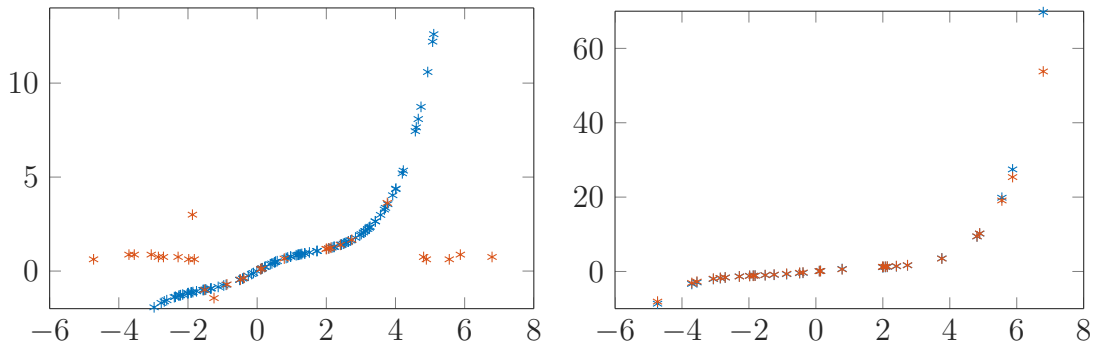

Figure 3: Interpolation results with a non orthogonal (left) and an orthogonal lagrange-basis (right).

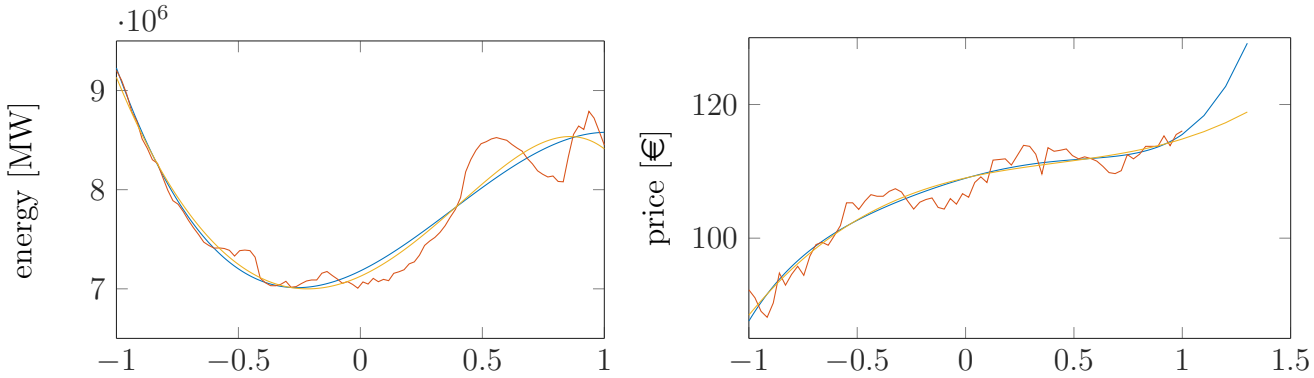

Figure 4: Interpolation of the energy transformation in Belgium on January first 2015 and interpolation and extrapolation of the *Anheuser-Busch InBev SA/NV* stock form January first until April tenth with outlook. Values obtained from matlab are schown in blue, values computed with fortran are given in yellow. Input values are shown in orange.

instability as the polynomials used earlier. For non-orthogonal polynomials results improve if a smaller degree and thus less data points are used. However this results in a loss of accuracy. It is important to note, that for interpolation in the core part of the function non orthogonal polynomials perform equally well their orthogonal counterparts. Orthogonalization is therefore only necessary in larger problems or when extrapolation is desired. However orthogonal polynomials have only been obtained to degree two in fortran and up to third degree in matlab. Higher degree polynomials are *somewhat* orthogonal with orthogonality errors well above machine precision. The obtained polynomials have been applied successfully to real world problems, with inter- and extrapolation using the classical least squares approach, results are shown in figure 4.

4 Conclusion

The goal of this assignment was to create software which is capable of reading in data in a given fixed format, decompose it and give approximations in certain points. This goal has been achieved. Numerous lines of code have been written delivering diverse functionalities like polynomial convolution, numerical integration or command line parsing. Orthogonal function representations can be computed up to degree two in fortran and degree three

in matlab. The requested projection algorithm does not work in the current version as higher order polynomials currently fail either the normality or orthogonality check depending on the order normalization or orthogonalization is done in the Gram-Schmidt module. All other requested functions are present, and have been carefully tested to the extend possible in the time available.