AGI: CONTINUITY IMPLEMENTATION

Mykola Rabchevskiy

The chapter "*AGI IN A WORLD OF TRAJECTORIES. How continuity affects AGI design*" discussed the need to represent quantities as functions of time. The video clip of the chapter "*AGI: STRUCTURING THE OBSERVABLE. How to detect unknown things and explain it*" demonstrates the use of such a representation in the problem of detecting structures in observable space. This chapter describes a practical implementation using ***C++***, the code for which is available at *https://github.com/mrabchevskiy/dynamic*.

The approximate time dependence is represented by a ***linear combination of the functions of the functional basis***, the combination coefficients of which are selected based on the requirement of the ***minimum standard deviation of the approximation*** from the known values at the measurement points. This problem is reduced to solving a system of linear equations with a dimension equal to the number of basis functions (the size of the functional basis). ***Extrapolation*** is reduced to using an approximation outside the time interval on which the data underlying the approximation are obtained.

The practical use of this relatively well-known approach is complicated because the system of equations to be solved may be ***ill-conditioned***, which leads to significant approximation errors or the practical impossibility of obtaining a solution. The implemented approach is described below, which allows avoiding this danger and thereby providing a reliable approximation in all situations. Those who are not interested in mathematical details can omit them and jump to the software code description.

MATHEMATICAL ASPECTS

The system of equations is characterized by a matrix of coefficients, which is ***symmetric and non-negative*** definite. The latter means that cases are possible when the matrix is ***degenerate***, and the system of equations has ***many solutions***. When the matrix is not entirely degenerate but ***poorly conditioned***, the only solution formally turns out to be ***unstable***: small inevitable measurement and computational errors lead to significant approximation errors. The situation is influenced by the choice of the functional basis and the way the points of measurement of the approximated dependence are distributed; therefore, it is impossible to choose a functional basis in advance that ensures the absence of the described difficulties.

The way out is to use a special method for solving a linear system based on the analysis of the ***eigenvalues and eigenvectors*** of the matrix of coefficients of the system of equations. First of all, the corresponding complete ***problem of the eigenvalues*** of the coefficient matrix is solved. A ***new functional basis is constructed*** from the selected subset of eigenvectors, which provides good conditioning (those eigenvectors are discarded that can enormously change the result with minor changes in the measured values).

Polynomials are used as elements of the functional basis. The choice of the powers of the independent variable as such functions contributes to the degeneracy of the approximation problem; therefore, a set of ***Chebyshev polynomials*** is used, mutually orthogonal on the interval [-1 .. +1], and the argument (time) is reduced to this interval by ***scaling and displacement***.

The sought approximation, being a linear combination of polynomial functions of the basis, is also a ***polynomial***; the result is a general polynomial of the corresponding order. The use of ***Horner's scheme*** in calculating the value of the approximation polynomial provides the minimum computational error and the maximum speed of this stage.

As a result, this approach allows one to construct an approximation in any situation, ***including those when the number of points with known values is less than the dimension of the functional basis*** (and, accordingly, the number of coefficients of the approximating polynomial).

SOURCE CODE

The template class ***Dynamic*** code that implements the idea of a regularly updated time function, which is built based on a series of measured values, is placed in the header file ***dynamic.h***. The scenario for using an instance of ***Dynamic*** is that after creating an object, the following operations are performed asynchronously in an arbitrary order:

- Replenishment of the list of samples "time-value pairs" (or clearing this list).
- Recalculation of the approximation function using the current list of samples.
- Calculating the value of the approximated/extrapolated value for an arbitrary time.

The ***Dynamic*** class is thread-safe; it is assumed that these operations can be performed by permanent services implemented as separate threads.

The template parameters are the functional basis's size and the approximated value type (***double*** by default).

The size of the internal list of samples is limited by the constructor's parameter value. The oldest one inserted earlier is deleted when a new sample is added to the list with the maximum possible number of samples. Once created, the object does not use memory allocation.

The second parameter of the constructor is the functional basis; The ***polynomial.h*** header contains the code of the template classes ***Polynomial*** and ***PolynomialBasis***, as well as constant objects of polynomial bases up to the fifth-order (having up to 6 polynomial coefficients, respectively), composed of Chebyshev polynomials. Header ***eigen.h*** contains code that implements linear algebra operations, ***heapsort.h*** implements sorting an array of objects on the place (no memory allocation), ***timer.h*** contains a class that measures time intervals, ***range.h*** **-** an iterator over a range of index values.

The ***dynamic.cpp*** file contains code that demonstrates the use of ***Dynamic*** objects and also acts as a unit test. Comments and texts in console output statements in the code describe the meaning of the actions to be performed, playing the role of "short manual". The console output of the test run is placed in the ***dynamic.log*** file.

The quality of extrapolation obviously depends on how far the extrapolation zone extends beyond the measurement interval. In this implementation, the length of the extrapolation zone is set by a hard-coded constant in ***dynamic.h*** and is equal to half the size of the measurement zone:

```
constexpr Real FACTOR{ 0.5 };
```

The tests performed by the demo program include the approximation of the values of the function specified by the polynomial (reproduced with an accuracy of calculation errors) and the approximation/extrapolation of the coordinates of a point moving along an arc of a circle; the output contains data on the magnitude of the error.

The code assumes the use of the ***C++-20*** standard and is compiled with ***GCC-10***.

Subscribe to AGI engineering

By Mykola Rabchevskiy · Launched 2 years ago

AGI: fundamentals, architecture, implementation, source code