

Re: how to evaluate the addition of millions of functions dynamically and efficiently?

Source: <http://sci.tech-archive.net/Archive/sci.math.num-analysis/2004-08/0213.html>

From: jim green (jjg_at_withheld.org.de)

Date: 08/16/04

Date: Mon, 16 Aug 2004 12:19:51 GMT

"networm" <networm8848@yahoo.com> writes:

- > *Is your algorithm supposed only to work only for regularly laid-out*
- > *functions? I mean for the support of functions, I can make them truncated*
- > *into rectangular-shaped finite support, even for infinite support function.*

Yes, the trick works best for

- translates of the same function with small support
- arranged on a regular grid (might be square, rectangular, triangular)

I'd recommend against truncating a function with infinite support -- this will lead to a discontinuity and so to a slow decay in the fourier transform. Instead (if possible) use a basis function which is compactly supported and smooth. Search for "Wendland" or "Lewitt" along with "radial basis functions".

- > *These functions are nicely behaved. So that's ok. But for the layout of the*
- > *center of these functions, for this function array, your dynamic local*
- > *summation only work for horizontally and vertically alligned function-center*
- > *array layouts?*

Not necessarily. You just need to be able to work out what the indices of the nearby points are **without** doing a test of every possible grid node. This can be done for a triangular grid, for example. If your grid nodes are genuinely "scattered" then the easy way wont work.

- > *How to decide if a point is covered by which overlapped neighboring*
- > *functions, for those irregular layout? I guess there should be some fast*
- > *algorithm and fast data structure for handling this in the literature?*

Possibly you could use a 2-stage method

- 1) find the neighbours for each grid node & store in a file
- 2) read neighbours into memory and for each node do the local search with these neighbours

sci.math.num-analysis: Re: how to evaluate the addition of millions of functions dynamically and efficiently?

Then 1) (which is non-trivial, but there is some code out there for doing it) has (if I remember correctly) complexity $O(N^2 \log(N))$, but 2) has $O(N)$, and 1) only need to be done once.

Note that this may require *lots* of memory. Work out how much you will need before you start to code!

-j