Green's Functions in many dimensions and multiwavelets

Python for a new class of fast algorithms

Fernando Pérez
fperez@colorado.edu

Gregory Beylkin, Vani Cheruvu
Robert Cramer, Lucas Monzón - CU Boulder
Martin Mohlenkamp - Ohio University
Robert Harrison, George Fann, Takeshi Yanai, Zhengting Gan - ORNL

Dept. of Applied Mathematics

CU BOULDER

SciPy'04
Caltech, Sept. 2 2004

Preliminaries

$$$ DOE, NSF, DARPA, ORNL.

A tale in 2 parts:

1. Some mathematics and physics:
   - Interesting ideas about a new class of algorithms.
   - Applicable to many fields.
   - I'll keep the techincal details to an absolute minimum.
   - Anyone interested in those, talk to me later.

2. Software: Python's strengths (and some weaknesses) for this problem.

Motivation: the 'curse of dimensionality'

A simple observation: numerical algorithms $(C = AB, y = Ax)$ in d physical dimensions scale like $\mathcal{O}(k^d)$. This is a problem, to put it mildly.

A typical example: Poisson's equation (electromagnetics, gravity, ...):

$$\nabla^2 \phi(r) = \rho(r)$$

A Green's function solution (free space, $d = 3$, ignore constants) :

$$\phi(r) = \int G(r - r')\rho(r')d^3r' = \int \frac{1}{|r - r'|}\rho(r')d^3r'.$$

If we discretize using a global basis, this becomes:

$$\phi_{mnp} = \sum_{m'n'p'=1}^{N} G_{mm',nn',pp'}\rho_{m'n'p'} \rightarrow \mathcal{O}(N^6)$$

Applying an integral kernel is a matrix-vector multiplication.

Can we do this efficiently for $d > 1$?

- Adaptive grid & a local basis of order k: $\mathcal{O}(N_{blk}k^6)$ [may be better than $\mathcal{O}(N^6)$]
- What if we could write:

$$G_{mm',nn',pp'} = \sum_{r=1}^{R} w_r F^r_{mm'} F^r_{nn'} F^r_{pp'}.$$

We could separate the different dimensions:

$$\phi_{mnp} = \sum_{r=1}^{R} w_r \sum_{m'} F^r_{mm'} \sum_{n'} F^r_{nn'} \sum_{p'} F^r_{pp'} \rho_{m'n'p'} \rightarrow \mathcal{O}(N_{blk}k^4)$$

The problem partially factorizes.

We can!

Key ideas:

1. Multiresolution analysis (wavelets): sparse matrix representations for a large class of kernels.
2. Separated representations: reduction of dimensionality cost.

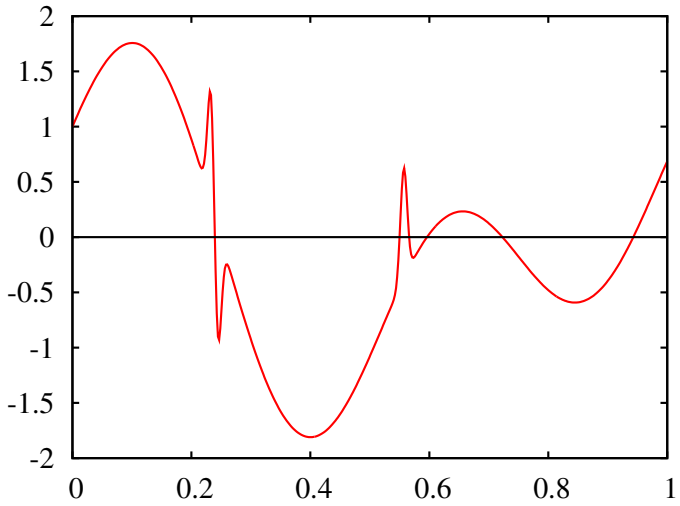
Python: ideal language for these algorithms, which rely heavily on sparse data structures (more than just matrices). Dictionaries to the rescue.

Some references

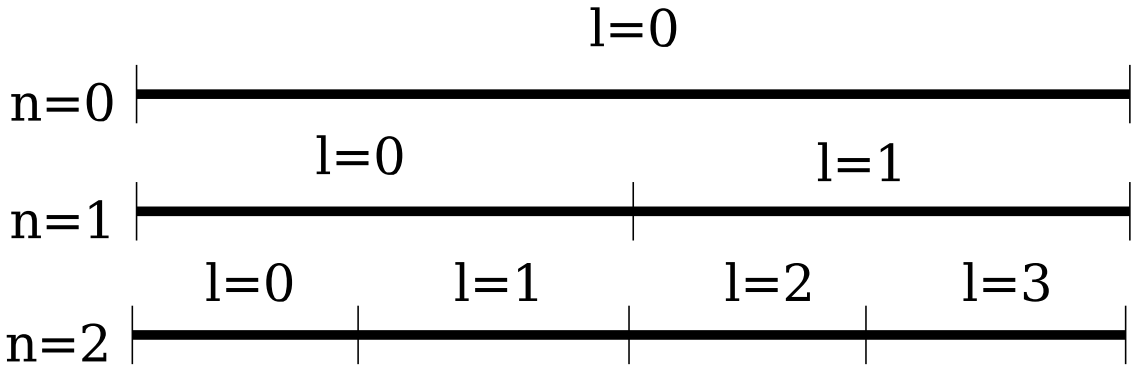
- Beylkin, Coifman, Rokhlin [Comm. Pure App. Math. **44**, 141-183 (1991)]
- Alpert [SIAM J. Math. Anal. **24**, 246-262 (1993)]
- Alpert, Beylkin, Gines, Vozovoi [J. Comp. Phys. **182**, 149-190 (2002)]
- Beylkin and Mohlenkamp [Proc. Nat. Acad. Sci. **99**, 10246-10251 (2002)]
- Beylkin and Monzón [App. Comp. Harmonic Analysis **12**, 332-373 (2002)]

Multiresolution Analysis - the bare basics

Imagine a simple signal you want to study:



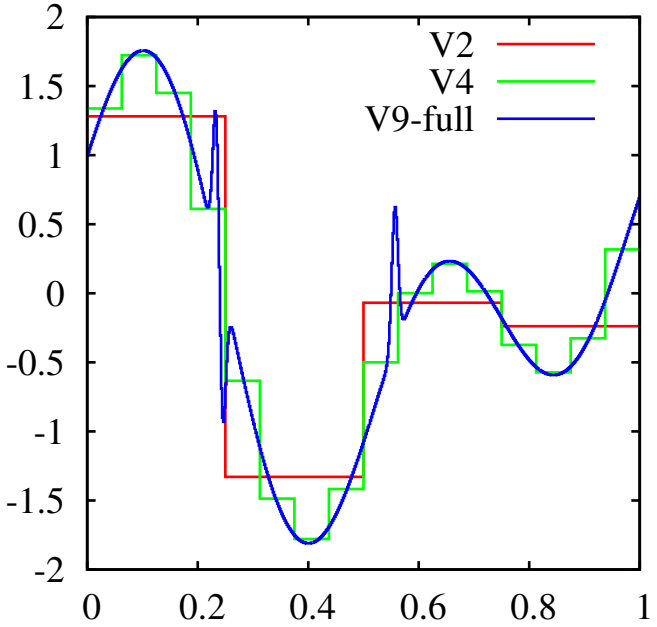
At each level n, divide the unit interval $[0, 1]$ into 2^n binary subintervals:

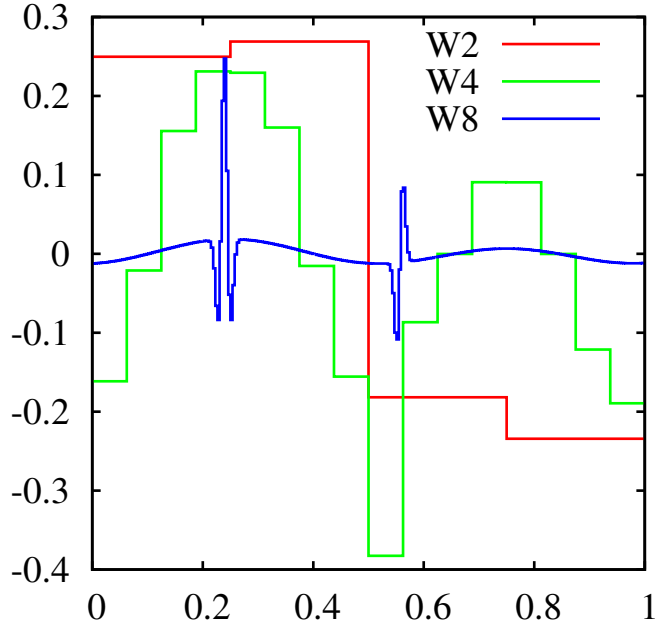


And compute:

- Average (s) values of function at level n : space V_n.
- Differences (d) between successive levels: space $W_n = V_{n+1} - V_n$.

The signal can studied (compressed, denoised, ...) from $\{V_0, W_0, W_1, \ldots\}$:



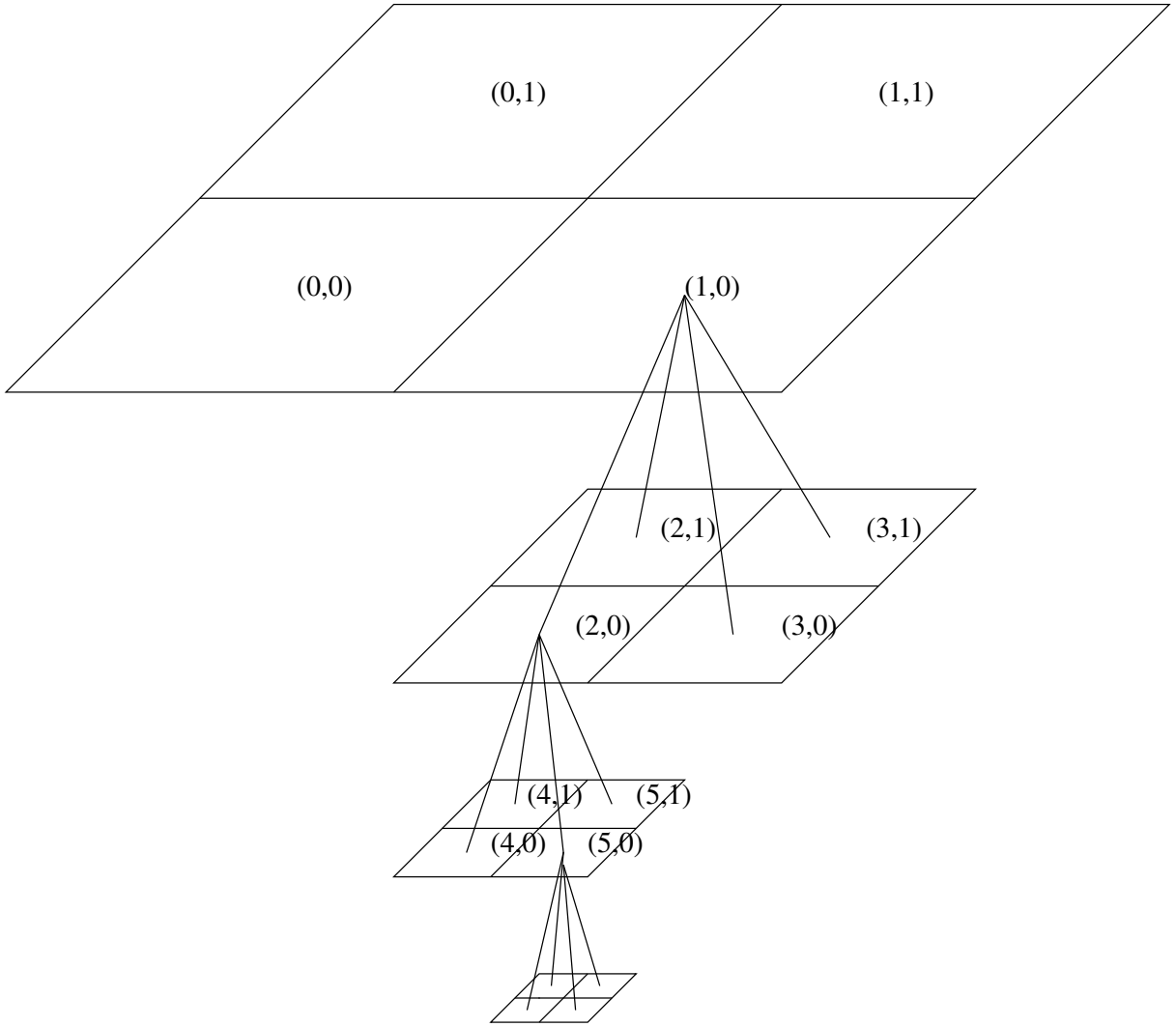


The d coefficients are small and localized around changes.

More complex bases (multiwavelets): N_{nod} coefficients per subinterval.

Adaptive subdivision of the unit interval in $\mathbb{R}^d$

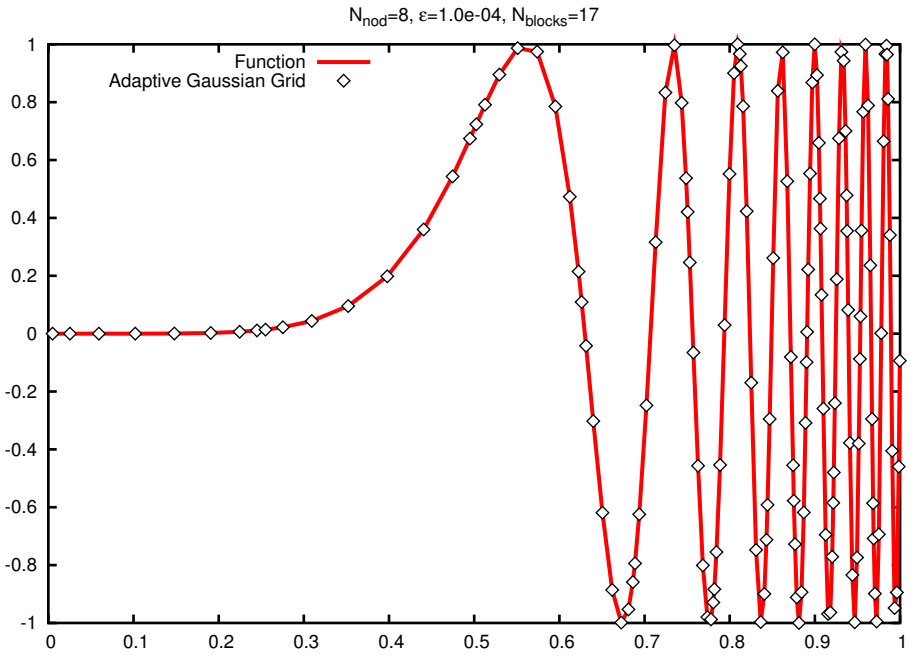
Simple recursive subdivision produces a d-binary tree on the unit interval, with N_{nod}^d coefficient blocks on the leaves:

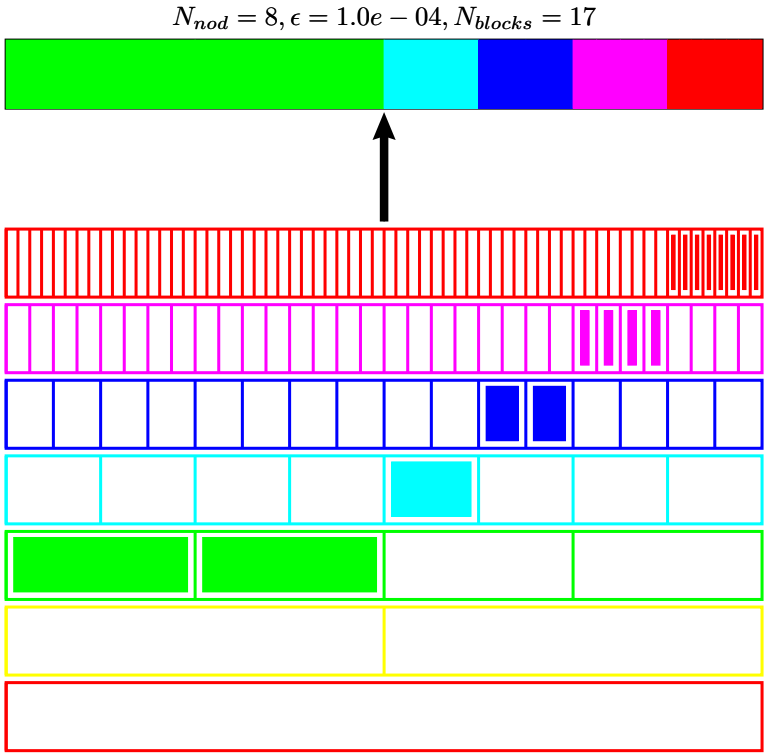


Decomposing functions with prescribed accuracy

A simple 1-d example, using $N_{nod} = 8$, $\epsilon = 10^{-4}$ for

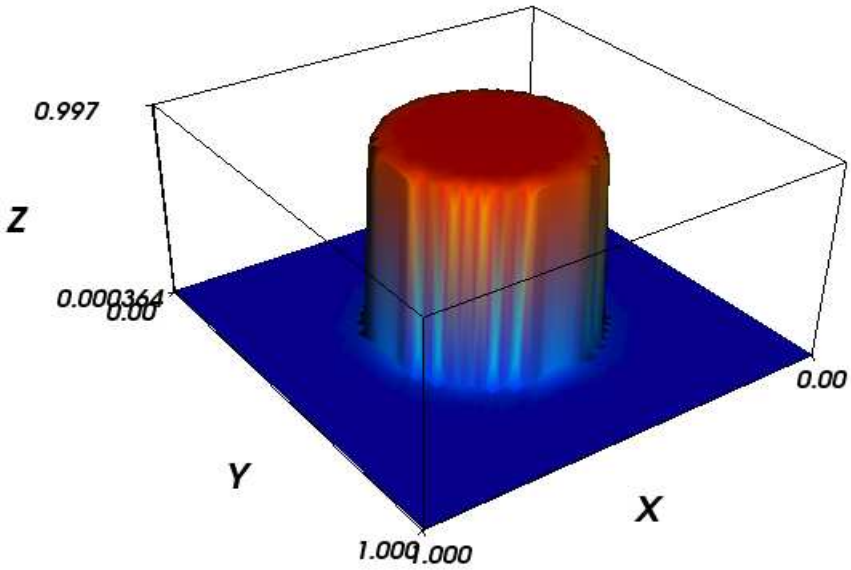
$$f(x) = \sin(16\pi x^6)$$



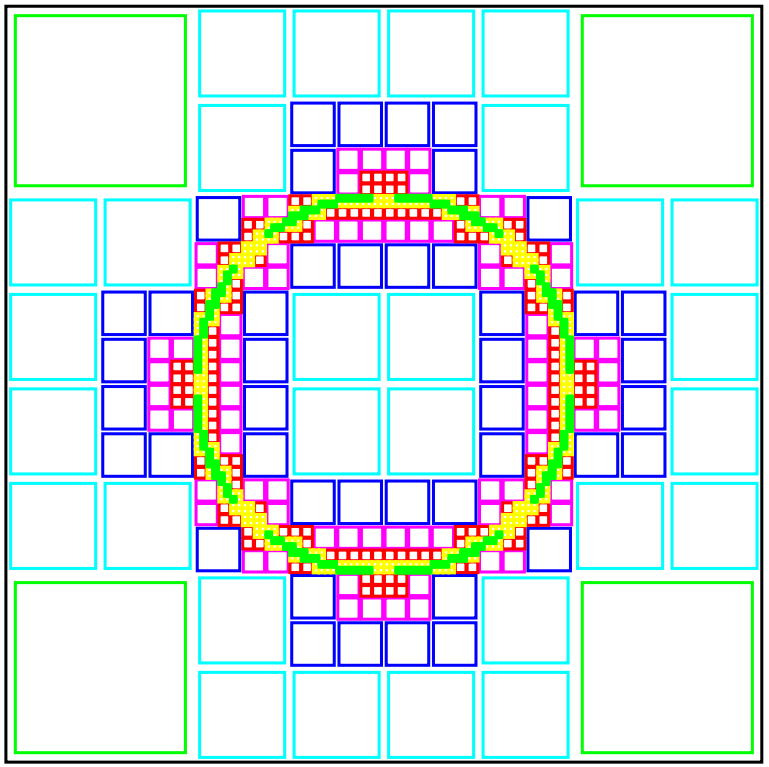


Resolving sharp discontinuities (a 2d example)

Consider a discontinuity along a circle (typical charge density for a 2d electrostatics problem).



$N_{nod} = 8, \epsilon = 1.0e-02, N_{blocks} = 1276$

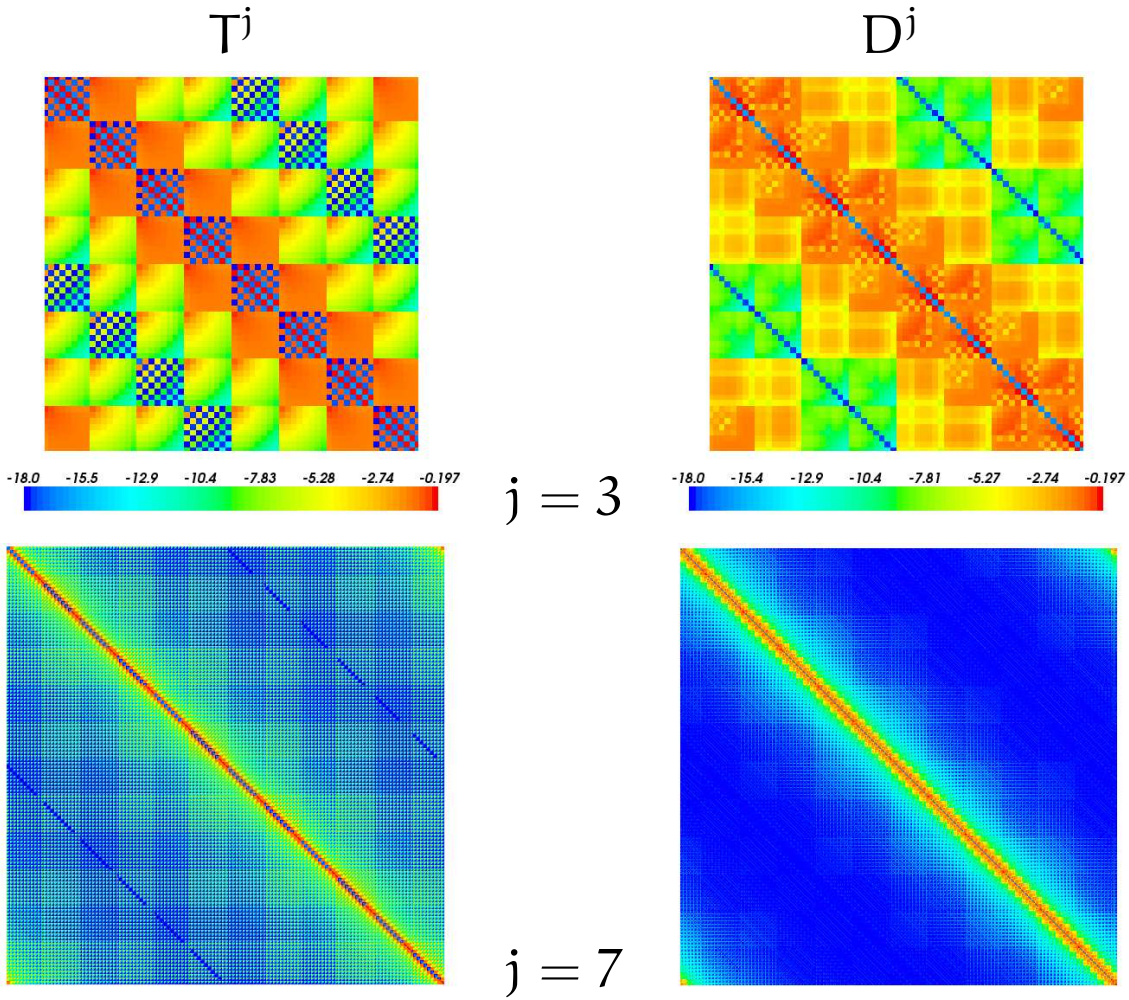


The trees require more blocks around the discontinuty, but not elsewhere.

Operators (1): sparse representations

Same idea, project the operator on each scale and use differences:

$$T^n = T^0 + (T^1 - T^0) + (T^2 - T^1) + \ldots = T^0 + \sum_{j=1}^{n} D^j.$$

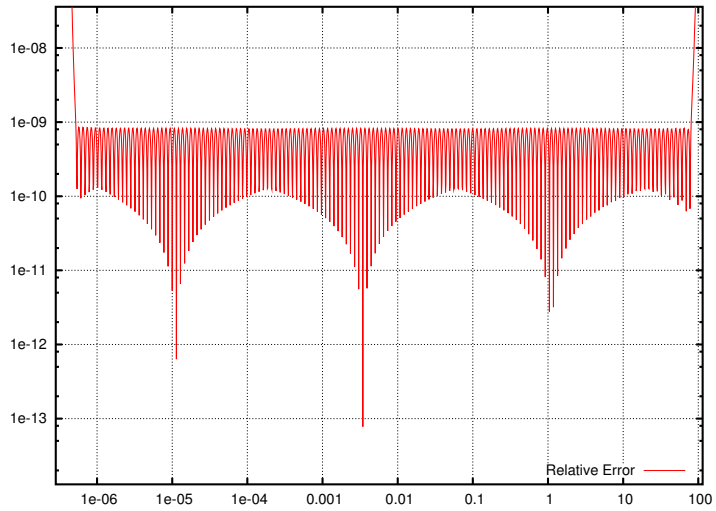


Operators (2): separation to break dimensions

It turns out we can approximate a wide class of functions as sums of Gaussians:

$$\frac{1}{\|r - r'\|} \approx \sum_{m=1}^{M} w_m e^{-\tau_m \|r-r'\|^2},$$

with controlled accuracy ϵ over a wide dynamic range [$M \approx \mathcal{O}(-\log \epsilon)$]:



This gives us the factorization we wanted for our kernel:

$$G_{ii'',jj'',kk''} = \sum_{m=1}^{M} w_m F^m_{ii'} F^m_{jj'} F^m_{kk'}.$$

A specific example

Let's solve Poisson's equation

$$\nabla^2 \phi(\mathbf{r}) = \rho(\mathbf{r}),$$

with

$$\rho(\mathbf{r};\alpha) = (6\alpha - 4\alpha^2 r^2) e^{-\alpha r^2} \Longrightarrow \phi(\mathbf{r};\alpha) = e^{-\alpha r^2}.$$

[DEMO]

Some comments:

- More optimizations coming down the pipe (structural)
- Rewrite *one* routine in C ($\sim$25 lines of Python in 3d)
- Very competitive with multigrid codes for high accuracy, where multigrid slows down dramatically.
- We think it's competitive with FMM codes (still testing).
- Python overhead measured as low as $\sim 1 - 2\%$.

Quantum Chemistry (Oak Ridge National Lab.)

- R. Harrison et. al. have implemented these ideas for electronic structure of many electron atoms and simple molecules.
- Multiresolution approach: complete elimination of basis set error.
- Systematic control of accuracy
- Work in progress to build $2e^-$ bases (requires 6-d codes)
- Parallelizing for the Cray X1 at ORNL

Math/Physics summary

- New class of algorithms, holds promise for attacking problems in high dimensions
- Applications exist: Poisson, Schrödinger, Navier-Stokes (prototype)
- Ideas applicable to many different fields (PDEs are everywhere)
- Once the algorithms are in place, writing codes for physical problems should be very easy (already true for the cases we have).
- Open questions remain (technical)

Python's benefits for this problem (1)

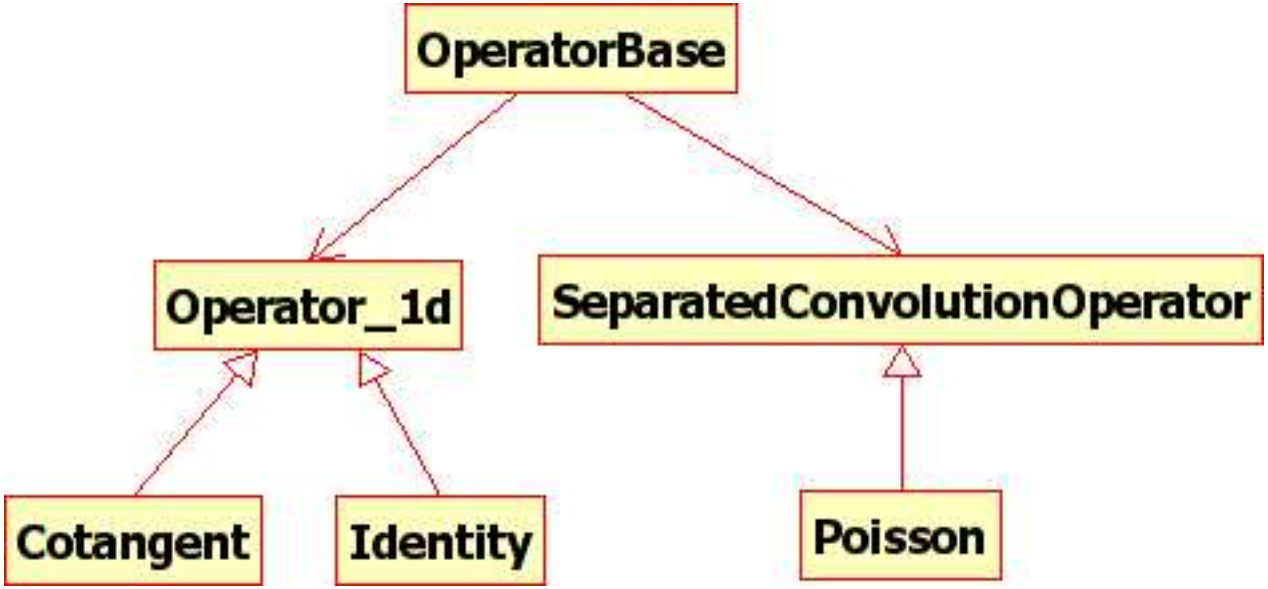
- Rich, flexible, well implemented native types: Dicts, Numeric arrays, Sets, Lists/Tuples
- F2PY, weave.inline(): painless access to Fortran and C/C++.
- Graphics tools:
  - Gnuplot
  - MayaVi
  - PyX
- Interactive work: IPython
- Object oriented (fits the problem domain like a glove)

Python's benefits for this problem (2)

- Function decomposition algorithm:
  - 1-6 d in a single classs (tiny amount of d-dep. code)
  - Trivial implementation
  - Performed better than the Fortran (3d only) it replaced: some algorithmic optimizations were much easier to see/implemement in Python
- Operator code:
  - 1d first, then jumped to 3d (skipped 2d).
  - We had to backtrack to 2d to compare with FMM literature.

Code structure

OO library:



"Smart" objects:

- `.info*()`, `.plot*()` methods [every figure in this talk was generated by a method of one of our objects]
- Tailored for interactive work: "live" debugging (IPython's `@run` and judicious use of `reload()`)

Not all is perfect in the land of the snake...

- Python is slow (and here it matters)
  - We know all the tricks, but we want to code even our loops in Python!
  - Where did Pat Miller's PyTRAN go?
  - Static typing? Dynamism is a killer in a loop where types never change.
  - Function call overhead is also a real problem for us.
- Numeric/Numarray situation is not good:
  - I *want* to use Numarray (clean design, nice docs, ...)
  - BUT: I may need $\mathcal{O}(10^6)$ array constructions (12×12 or $12 \times 12 \times 12$). The Numarray overhead would be too much here.
  - I'm NOT blaming the Numarray team!
- The GIL (Python's Global Interpreter Lock): a real problem for parallelization (shared memory, not MPI). This is pushing us to C++.
- Scipy is underused/underappreciated: docs problem (I think). This is an area where the community can help (T. Oliphant and Enthought can only do so much).

A few useful URLs

- SciPy: http://scipy.org
- F2Py: http://cens.ioc.ee/projects/f2py2e/
  Python access to FORTRAN codes.
- SWIG: http://swig.org
  Python access to C/C++ codes.
- IPython: http://ipython.scipy.org
  Improved interactive shell.
- Gnuplot.py: http://gnuplot-py.sourceforge.net
- Matplotlib: http://matplotlib.sourceforge.net
  High quality 2d plotting
- MayaVi: http://mayavi.sourceforge.net
  3d visualization software, VTK based (fast, flexible and powerful).
- PyX: http://pyx.sourceforge.net
  Programmatic PostScript, TEX and LATEX generation.