Efficient Bessel Decomposition toolbox

Martin Averseng

September 17, 2020

Abstract

This toolbox implements the algorithm described in my paper "Discrete convolution in $\mathbb{R}^2$ with radial kernels using non-uniform fast Fourier transform with non-equispaced frequencies", published in Numerical Algorithms in 2019.

The method is designed to compute fast approximations of vectors q which entries are given by

$$q_k = \sum_{l=1}^{N_y} G(X_k - Y_l) f_l, \quad k = 1, \cdots, N_x$$

or

$$q_k = \sum_{l=1}^{N_y} \nabla G(X_k - Y_l) f_l, \quad k = 1, \cdots, N_x$$

where G is a radial function, i.e. $G(x) = g(|x|)$ for some function g, X and Y are two coulds or N_x and N_y points in $\mathbb{R}^2$, and f is a complex vector.

To test it, you can directly run Demo.m, or DemoGrad. Here follows a more detailed description of the algorithm and a tutorial.

1 Description of the algorithm

- The method first decomposes G in finite Bessel series

$$G(r) = \sum_{p=1}^{P} \alpha_p J_0(\rho_p r)$$

where J_0 is the Bessel function of first kind, ρ is the sequence of its positive roots, and α are called the EBD coefficients of G. The coefficients are chosen as the minimizers of the Sobolev H_0^1 norm of the error in this approximation on a ring $r_{min} < r < r_{max}$ where r_min is a cutoff parameter and r_max is the greatest distance occurring between two points X_k and Y_l. The method takes the paramter $a = r_{min}/r_{max}$ as an input.

- Second, each $J_0(\rho_p r)$ is approximated by a sum of complex exponentials via

$$J_0(\rho_p|x|) = \frac{1}{M_p}\sum_{m=1}^{M_p} e^{i\rho_p \xi_m^p \cdot x}$$

where $\xi_m^p = e^{i2m\pi/M_p}$. This is the trapezoidal rule applied to the formula

$$J_0(|x|) = \int_{\partial B} e^{ix\cdot\xi} d\xi$$

where the integration takes place on the boundary of the unit disk B in $\mathbb{R}^2$.

- Combining these two steps, we obtain an approximation for G of the form

$$G(x) \approx \sum_{\nu=1}^{N_\xi} \hat{\omega}_\nu e^{i\xi_\nu \cdot x}$$

valid for $r_{min} \leq |x| \leq r_{max}$. If this is replaced in the expression of q_k, we see that q can be approximated by non-uniform Fourier transform for any vector f. The interactions $|X_k - Y_l| < r_{min}$ where the approximation is not valid are corrected by a sparse matrix product.

The code is in Matlab language. The implementation of the NUFFT is borrowed from Leslie Greengard, June-Yub Lee and Zydrunas Gimbutas (see license file in the libGgNufft2D folder). The ideas come from a similar method in 3D called Sparse Cardinal Sine Decomposition, developped by François Alouges and Matthieu Aussal, also published in Numerical Algorithms.

2 Tutorial

2.1 Simple EBD

- Create the arrays X and Y of sizes Nx x 2 and Ny x 2 (points in $\mathbb{R}^2$).
- Create a kernel by calling

```
G = Kernel(fun,der);
```

where `der` is an anonymous function of your choice and `der` is an anonymous function repesenting the derivative of `der`. For example,

```
G = Kernel(@(x)(1./x),@(x)(-1./x.^2));
```

Be aware that the anonymous functions provided in argument must accept arrays as input. For some classical kernels, the methods have been optimized. You can create one of those special kernels using the Kernel library:

```
G = LogKernel; % represents G(x) = log(x)
G = ThinPlate(a,b); % represents G(x) = a*x^2*log(b*x)
```

(and others, see folder Kernels).

- Define the tolerance in the error of approximation. The method guarantees that the Bessel decomposition of G in the ring is accurate at the tolerance level. This implies that the error on an entry of q_k is at most `tol*norm(q,1)`.
- Define the parameter a (the ratio between r_{min} and r_{max}.) It is roughly the proportion of interactions that will be computed exactly. There is an optimal value of a for which the evaluation of the convolution is the fastest, but it is not possible to know it in advance. However, when X and Y are uniformly distributed on a disk, the optimal a is of the order $\frac{1}{(N_x N_y)^{1/4}}$, and if they are uniformly distributed on a curve, of the order $\frac{1}{(N_x N_y)^{1/3}}$. If a is large, a lot of interactions are computed exactly while the Bessel decomposition will have only a few terms. If a is small, the opposite will happen.
- You can now call

```
[onlineEBD, rq, loc] = offlineEBD(G,X,Y,a,tol);
```

The variable `onlineEBD` contains a function handle. If f is a vector with length equal to `size(Y,1)`,

```
q = onlineEBD(f);
```

returns the approximation of the convolution. The variable `rq` contains a `RadialQuadrature` object. You can visualize a representation of the radial approximation with `rq.show()` and also check all its properties (including coefficients, frequencies used, accuracy,...). Finally, `loc` is the sparse local correction matrix (used inside `onlineEBD`).

2.2 Derivative EBD

If you rather want to compute the vector

$$[q_1(k), q_2(k)] = \sum_{l=1}^{N_y} \nabla G(Y_l - X_k) f_l$$

where for a point $x \in \mathbb{R}^2$,

$$\nabla G(x) = g'(|x|)\frac{x}{|x|}$$

where $G(x) = g(|x|)$, you may call `offline_dEBD` instead of `offlineEBD`. The syntax is

```
[MVx,MVy,rq,loc] = offline_dEBD(G,X,Y,a,tol);
```

where, for example, `MVx` contains the function handle such that

```
q1 = MVx(f);
```

is the component q_1 of the convolution, and `loc` is a 2×1 cell containing the sparse correction matrices for each component