Generalized sampling in Julia

Jacobsen, Christian Robert Dahl; Nielsen, Morten; Rasmussen, Morten Grud

Published in:
Journal of Open Research Software

DOI (link to publication from Publisher):
10.5334/jors.157

Publication date:
2017

Document Version
Publisher's PDF, also known as Version of record

Journal of
open research software

Jacobsen, R D et al 2017 Generalized Sampling in Julia. *Journal of Open Research Software*, 5: 12, DOI: https://doi.org/10.5334/jors.157

SOFTWARE METAPAPER

Generalized Sampling in Julia

Robert Dahl Jacobsen, Morten Nielsen and Morten Grud Rasmussen

Department of Mathematical Sciences, Aalborg University, Fredrik Bajers Vej 7G, DK

Corresponding author: Robert Dahl Jacobsen
(robert@math.aau.dk)

Generalized sampling is a numerically stable framework for obtaining reconstructions of signals in different bases and frames from their samples. For example, one can use wavelet bases for reconstruction given frequency measurements.

In this paper, we will introduce a carefully documented toolbox for performing generalized sampling in Julia. Julia is a new language for technical computing with focus on performance, which is ideally suited to handle the large size problems often encountered in generalized sampling. The toolbox provides specialized solutions for the setup of Fourier bases and wavelets.

The performance of the toolbox is compared to existing implementations of generalized sampling in MATLAB.

Keywords: Generalized Sampling; Julia; Fourier basis; Wavelets
Funding Statement: Supported by the Danish Council for Independent Research | Natural Sciences, grant 12-124675, "Mathematical and Statistical Analysis of Spatial Data". RDJ is supported by the Centre for Stochastic Geometry and Advanced Bioimaging, funded by a grant (8721) from the Villum Foundation.

(1) Overview

Introduction

Generalized sampling [1, 2] is a framework for estimating representations of functions in different bases and frames in a numerically stable manner. This can for example be relevant in processing MRI images, where hardware often enforces initial frequency measurements, but where wavelets may be better suited for representing the final image. Simply using a discrete inverse Fourier transform often results in the well-known Gibbs phenomenon, with unwanted oscillating behavior in the reconstructed image. This paper documents a toolbox for performing generalized sampling in Julia [3].

The theory of generalized sampling does not restrict the type of bases to consider, but the applications have focused on Fourier bases and multiscale representations like wavelets.

Mathematically, samples of a function f in a Hilbert space $\mathcal{H}$ with respect to a sample basis $\{s_n\}_{n\in\mathbb{N}}$ consists of inner products $\{\langle f, s_n\rangle\}_{n\in\mathbb{N}}$. In generalized sampling we want to use these samples to estimate the inner products $\{\langle f, r_n\rangle\}_{n\in\mathbb{N}}$, where $\{r_n\}_{n\in\mathbb{N}}$ is another basis for $\mathcal{H}$. The basis $\{r_n\}_{n\in\mathbb{N}}$ is used for reconstructing f. In practice we only consider a finite number of sampling and reconstruction functions, i.e., we have access to the samples $w_m^s = \langle f, s_m\rangle$, $1 \le m \le M$. From these samples we estimate the coefficients in the reconstruction basis, $\tilde{w}_n^r \approx w_n^r = \langle f, r_n\rangle$, $1 \le n \le N$, which are used to compute an approximation of f,

$$\tilde{f}_{N,M} = \sum_{n=1}^{N} \tilde{w}_n^r r_n.$$

An especific example that will be used later is where $\langle f, s_m\rangle$ represent frequency samples of f and the reconstruction basis $\{r_n\}_{n=1}^N$ is given by translates of a Daubechies scaling function ϕ yielding a system on the form

$$\phi_{J,K}(x) = 2^{J/2}\phi(2^J x - k), \quad -2^{J-1} \le k < 2^{J-1},$$

that has been adapted to the interval $[-\frac{1}{2}, \frac{1}{2}]$ following [4]). The simplest Daubechies scaling function is the Haar scaling function given by

$$\phi(x) = \begin{cases} 1, & x \in [-\frac{1}{2}, \frac{1}{2}] \\ 0, & \text{otherwise.} \end{cases}$$

Other Daubechies scaling functions are at least continuous, but cannot be written in closed form in terms of elementary functions.

The actual computation of the reconstruction coefficients $\tilde{\boldsymbol{w}}^r = \{\tilde{w}_n^r\}_{n=1}^N$ is performed by solving a least squares problem. The infinite change of basis matrix between the sampling and reconstruction subspaces has (i, j)'th entry (r_j, s_i). We consider a finite $M \times N$ section of this matrix, denoted by T. The reconstruction coefficients are computed as the least squares solution

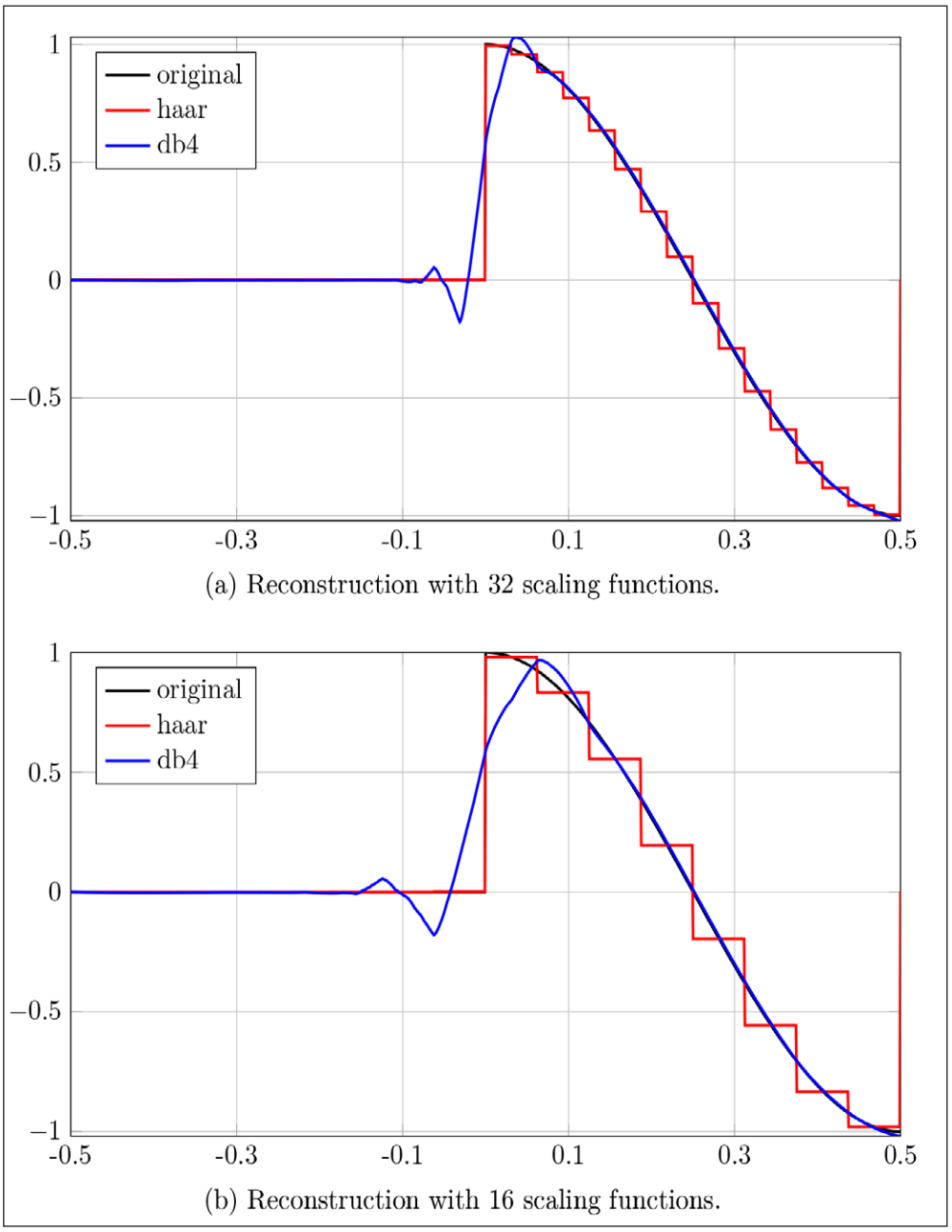

Figure 1: A truncated cosine and approximations with Haar and Daubechies 4 scaling functions using different scales. It may be difficult to see the graph of the original truncated cosine.

$$\tilde{\boldsymbol{w}}^r = \arg\min\left\{\left\|T\boldsymbol{x} - \boldsymbol{w}^s\right\|_2 \,\middle|\, \boldsymbol{x} \in \mathrm{C}^N\right\}. \tag{1}$$

For large M and N, e.g. of the size used in 2D examples with images, T is not accessible as a stored matrix. Instead we need to work with T as an operator, i.e., store a representation that allow us to compute products with T and T^* and thereby solving (1) with e.g. a conjugate gradient algorithm [6, p. 637].

Examples

We have two goals with the GeneralizedSampling package: It should be fast and easy to use. We have therefore put effort into providing only a few necessary high-level functions and hiding the lower level details. In essence, we need to compute a representation of a change of basis matrix T and solve least squares problems like (1) with this representation.

An example included in the package is the reconstruction of a truncated cosine (with inspiration from [5]). The results are seen in **Fig. 1** using a different number of scaling functions to illustrate how this affects the reconstruction. In this example the higher order, continuous Daubechies scaling functions are not well suited to represent the discontinuity.

A reconstruction like this can be computed in only a few lines of code once the frequency measurements are available. For the truncated cosine we have a closed-form expression for the Fourier transform:

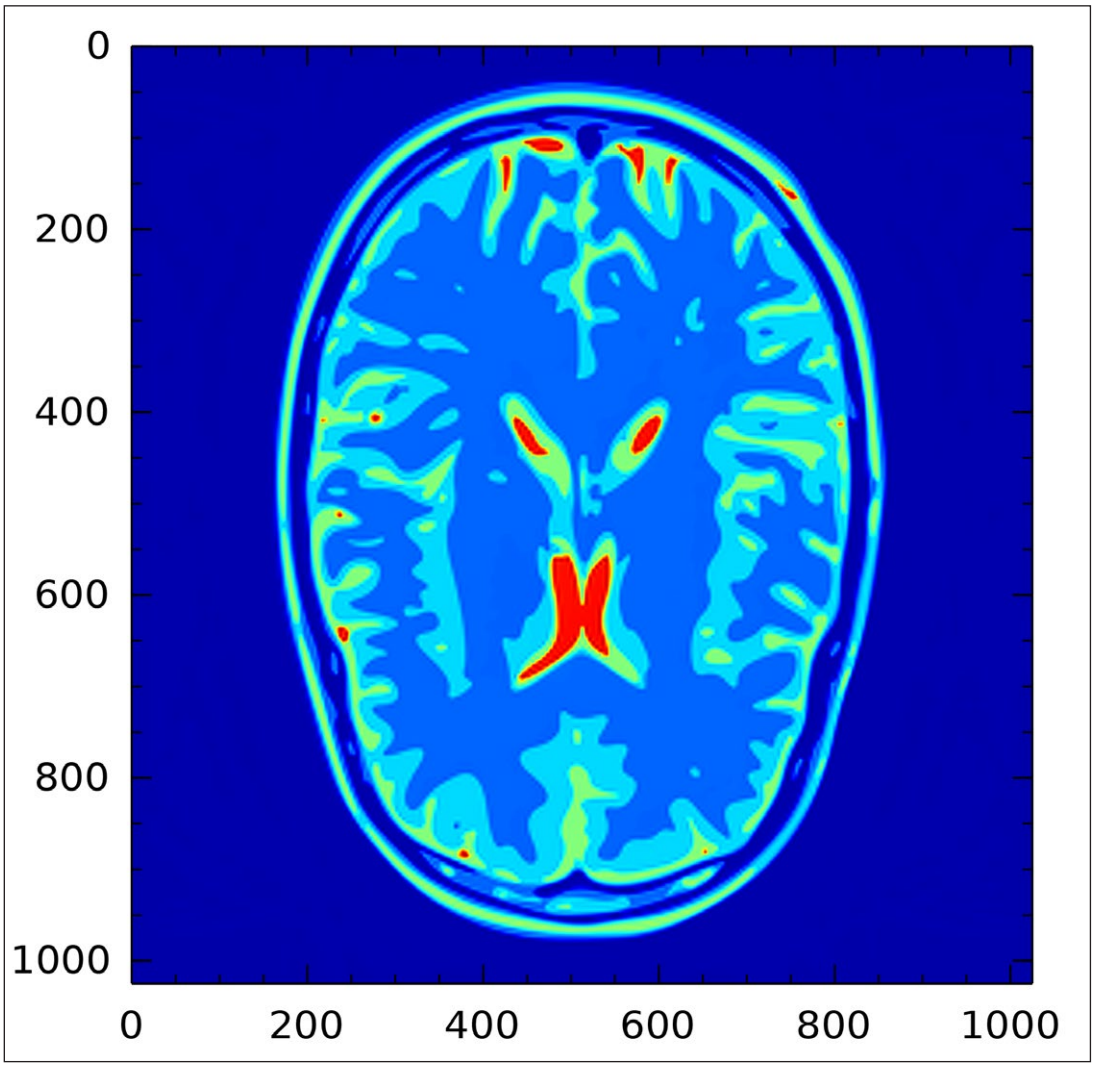

Figure 2: From 512 × 512 frequency measurements (not shown) 256 × 256 Daubechies 4 scaling functions are computed and used to produce this image. The image is evaluated at scale 10, i.e., in 1024 × 1024 points.

$$f(x) = \begin{cases} 0, & -\frac{1}{2} \le x < 0, \\ \cos(2\pi x) & 0 \le x \le \frac{1}{2}, \end{cases}$$

$$\hat{f}(\xi) = \begin{cases} \dfrac{i\xi(1+\exp(-\pi i\xi))}{2\pi(1-\xi^2)}, & |\xi| \neq 1, \\ \dfrac{1}{4}, & |\xi| = 1. \end{cases}$$

In Julia this can be implemented in the following way with a ternary operator (the line break is only for typographical purposes):

```
julia > ftcos (xi) = ( abs(xi) == 1 ? 0.25 :
im*xi *(1 + exp (-pi*xi*im ))/(2* pi *(1 - xi ^2)) )
julia > @vectorize_1arg Float64 ftcos
```

With the Fourier transform defined the reconstruction can be computed as follows:

```
julia > using GeneralizedSampling
julia > xi = grid (128 , 0.5)
julia > fhat = ftcos (xi)
julia > T = Freq2Wave (xi , " haar ", 6)
julia > wcoef = T \ fhat
julia > using IntervalWavelets
julia > x, y = weval ( real ( wcoef ), " haar ", 10)
```

To compute the reconstruction with the Daubechies 4 scaling functions, one simply has to replace "`haar`" with "`db4`".

As an example of reconstruction of 2D functions/images, consider reconstruction of a simulated brain made with the Matlab [8] software released along with [7] (available at http://bigwww.epfl.ch/algorithms/mriphantom). Here we do not have access to a "ground truth", but only the frequency measurements. From 512^2 frequency measurements we reconstruct 256^2 Daubechies 4 scaling functions and the result is seen in **Fig. 2**.

Runtime and technical comparison

Julia is a dynamic language with a fast JIT compiler. A consequence is that a function is compiled the first time it is called, causing an overhead in terms of time and memory. All subsequent calls are, however, without this compiling overhead. The runtimes reported in this section are *not* for the first run.

The examples were carried out on a normal laptop (2.60 GHz Intel Core i7, 8 GB RAM) running GNU/Linux and are summarized in **Table 1**. The background

			init		sol		Iter	
Problem	**Size**	**Program**	**time**	**mem**	**time**	**mem**	***n***	***t/n***
Uniform 1D	8192×4096	Matlab	15.0	901	0.13	37	9	0.01
		Julia	0.34	6.8	0.04	0.7	12	0.003
Jitter 1D	5463×2048	Matlab	10.0	627	0.28	59	20	0.13
		Julia	0.23	4.0	0.08	0.4	20	0.004
Uniform 2D	$512^2 \times 256^2$	Matlab	0.96	59	5.2	1507	9	0.58
		Julia	0.15	146	17.6	17	16	1.10
Jitter 2D	$26\,244 \times 32^2$	Matlab	104.8	6377	8.3	1270	50	0.17
		Julia	2.4	165	2.4	1.3	18	0.13
Spiral	$27\,681 \times 32^2$	Matlab	107.1	6670	3.7	479	17	0.21
		Julia	2.8	261	2.2	1.4	16	0.14

Table 1: Runtime comparisons with the Matlab implementation from [5] for reconstruction with the Daubechies 4 scaling functions. "Size" refers to the change of basis matrix; "init" is the computation of the change of basis matrix; "sol" is the computation of the solution; "Iter." is the number of iterations by the iterative solver; t/n is the time per iteration in the solver. Time is measured in seconds and "mem" is the memory allocation in megabytes.

for the experiments are as follows: We compare the GeneralizedSampling package with the Matlab code released with [5] (available at http://www.damtp.cam.ac.uk/research/afha/code). To the best of our knowledge this is the only other publicly available software for generalized sampling. In this connection two comparisons are relevant: Computing the representation of the change of matrix (initialization/"init") and using this representation to compute the least squares solution (solution/"sol"). In both cases the initialization step is fast for Haar scaling functions: All Fourier transforms have simple, closed-form expressions that are easily vectorized. For higher order Daubechies scaling functions all computations rely on iterative procedures.

In both packages the solution step is based on a conjugate gradient-like algorithm, where the computational cost is dominated by multiplication with the change of basis matrix and its adjoint. In Matlab the built-in `lsqr` function is used and in Julia a custom implementation of the conjugate gradient algorithm is used.

In GeneralizedSampling we have relied on Julia's ability to write functions that modify their arguments in-place to drastically reduce the memory consumption in an iterative algorithm like conjugate gradients. Julia's `@time` macro and the benchmarking mentioned below makes it easy to estimate the memory allocation of a function. In Matlab we have used an undocumented feature for the profiler, namely `profile -memory on` to measure the allocated memory.

Especially for fast runtimes it is not accurate to rely on timing a single run of a function (using e.g. `tic` and `toc` in Matlab). In Matlab the times are obtained with the built-in `timeit` function and in Julia we use the benchmark package `BenchmarkTools` available at https://github.com/JuliaCI/BenchmarkTools.jl. For small problems the speed of the Julia and Matlab code are comparable, but for large problems the Julia code is significantly faster. The one exception is for the "Uniform 2D" example: The Julia package is using a more general algorithm for non-equidistant fast Fourier transforms, whereas the Matlab package is considering a special case where the standard fast Fourier transform is applicable.

The total memory allocation of the solver is much smaller in the Julia code than in the Matlab code. In most cases, the memory allocation of the initialization code is smaller the Julia code than in the Matlab code, although this is probably not as important since it is only performed once.

Note that the "sol" times and number of iterations are not directly comparable, since the stopping criteria for the least squares solver may be different and [5] use $L^2([0, 1]^d)$ as the reconstruction space, whereas we use $L^2([-\frac{1}{2}, \frac{1}{2}]^d)$. But the time per iteration ("t/n") is comparable.

Quality control

An extensive automated test suite has been developed along with the main code and is tested with Travis CI and AppVeyor, allowing tests to be performed on both Unix-like platforms and Windows. All public functions have docstrings and a detailed documentation is included with the package in reStructuredText that can be exported to a multitude of formats. An HTML version of the documentation is hosted on Read the Docs. All details and links are available on the Generalized-Sampling package's GitHub page.

Furthermore, all examples have been tested by all authors for bugs and readability.

(2) Availability

Operating system

Any system capable of running Julia version 0.4 or higher.

Programming language

The GeneralizedSampling package has been tested with Julia version 0.4 and 0.5.

Additional system requirements
None.

Dependencies
The GeneralizedSampling package depends on the following Julia packages:

- NFFT (version 0.1.4)
- Wavelets (version 0.5.1)
- ArrayViews (version 0.6.4)
- VoronoiCells (version 0.1.3)
- IntervalWavelets (version 0.0.4)

List of contributors
- Jacobsen, Robert Dahl (Aalborg University) – Development.
- Nielsen, Morten (Aalborg University) – Testing and theoretical discussions.
- Rasmussen, Morten Grud (Aalborg University) – Testing and theoretical discussions.

Software location
Archive
Name: Zenodo
Persistent identifier: https://doi.org/10.5281/zenodo.168555
Licence: MIT "Expat"
Publisher: Robert Dahl Jacobsen
Version published: 0.0.5
Date published: 22/12/2016

Code repository
Name: GitHub
Persistent identifier: https://github.com/robertdj/GeneralizedSampling.jl
Licence: MIT "Expat"
Date published: 22/12/2016

Language
English.

(3) Reuse potential
The theory of generalized sampling is very generic with possible applications in many different fields, e.g. where the aquisition of samples are fixed by physical restrictions. The GeneralizedSampling package is easy to use and and does not require users to purchase additional software.

Some parts of the software are easily extended: If one wishes to change representation between a pair other than the Fourier and wavelet bases, this can easily be achieved by adding appropriate functions for computing representations of the appropriate change of basis matrix and for performing multiplication with this matrix. It is possible to contribute to the package through the GitHub interface.

Competing Interests
The authors have no competing interests to declare.

References

1. **Adcock, B** and **Hansen, A C** 2012 "A Generalized Sampling Theorem for Stable Reconstructions in Arbitrary Bases". In: *Journal of Fourier Analysis and Applications,* pp. 685–716. DOI: https://doi.org/10.1007/s00041-012-9221-x
2. **Adcock, B, Hansen, A C** and **Poon C** 2014 "On optimal wavelet reconstructions from Fourier samples: Linearity and universality of the stable sampling rate". In: *Applied and Computational Harmonic Analysis* 36.3 (May), pp. 387–415. DOI: https://doi.org/10.1016/j.acha.2013.07.001
3. **Bezanson, J, Edelman, A, Karpinski, S** and **Shah, V B** 2014 *Julia: A fresh approach to numerical computing.* SIAM Review, 59:65–98, doi:10.1137/141000671
4. **Cohen, A, Daubechies, I** and **Vial, P** 1993 "Wavelets on the Interval and Fast Wavelet Transforms". In: *Applied and Computational Harmonic Analysis* 1.1 (Dec), pp. 54–81. DOI: https://doi.org/10.1006/acha.1993.1005
5. **Gataric, M** and **Poon, C** 2016 "A practical guide to the recovery of wavelet coefficients from Fourier measurements". In: *SIAM Journal on Scientific Computing* 38.2, A1075–A1099. DOI: https://doi.org/10.1137/15M1018630
6. **Gene, H** 2013 Golub and Charles F. Van Loan. *Matrix Computations.* 4th ed. Baltimore: Johns Hopkins University Press,
7. **Guerquin-Kern, M, Lejeune, L Pruessmann, K P** and **Unser, M** 2012 "Realistic Analytical Phantoms for Parallel Magnetic Resonance Imaging." In: *IEEE Transactions on Medical Imaging* 31 (Mar), pp. 626–636. DOI: https://doi.org/10.1109/TMI.2011.2174158
8. **MATLAB Version R** 2016a (9.0.0.341360). Natick, Massachusetts, United States: The MathWorks Inc., 2016.

How to cite this article: Jacobsen, R D, Nielsen, M and Rasmussen, M G 2017 Generalized Sampling in Julia. *Journal of Open Research Software,* 5: 12, DOI: https://doi.org/10.5334/jors.157

Submitted: 29 November 2016 **Accepted:** 23 March 2017 **Published:** 20 April 2017

]u[*Journal of Open Research Software* is a peer-reviewed open access journal published by Ubiquity Press

OPEN ACCESS