Recursive filtering 2D Tikhonov regularization

Hermes H. Ferreira
Instituto de Informática – UFRGS
Porto Alegre, Brazil
hermes.ferreira@inf.ufrgs.br

Eduardo S. L. Gastal
Instituto de Informática – UFRGS
Porto Alegre, Brazil
eslgastal@inf.ufrgs.br

***Abstract*—In this work we describe an implementation of the 2D Tikhonov regularization filter which scales linearly with the input signal's size. We propose a novel algorithm to decompose the filter's 2D kernel as a sum of axis-aligned Gaussians. Our algorithm uses symmetries of the kernel to provide a fast computation of the Gaussian decomposition in the frequency domain, where the 2D Tikhonov kernel has a closed-form expression. The convolution with each Gaussian is then computed using linear-time separable recursive filtering. This way, a fast solution to the 2D Tikhonov regularization problem is obtained.**

I. Introduction

The Tikhonov regularization method arises in many different contexts. It was developed by Tikhonov and Arsenin [17] for solving ill-posed problems, and was also discovered independently by Hoerl and Kennard [10], who introduced it as Ridge regression in the context of statistics. The method is particularly useful for solving inverse problems, even non-linear ones. Such problems arise, for example, in Geophysics [9], Economics [6], Image Processing [1], and Tomography [18].

A regularized problem is usually stated as $\text{argmin}_x \|Ax - b\|^2 + \|\Gamma x\|^2$, which means solving $Ax \approx b$ with a regularization term Γx. In the work of Romano et al. [16], the problem is split in two steps, where one of the steps is applying a denoising engine $h(y)$ to an intermediate solution y. In our context $h(y) = \text{argmin}_x \|x-y\|^2 + \|\Gamma x\|^2$, which we will refer to as Tikhonov filter or regularization (Fig. 1). This particular formulation is also used by Nielsen et al. [15], who describe the relationship between the 1D Tikhonov problem (*i.e.*, where x is the sampling of a function from $\mathbb{R}$ to $\mathbb{R}$) and recursive filtering, when Γ is a scaled 1D derivative operator.

In this work we develop a fast linear-time algorithm ($O(N)$ in the input size N) for applying the denoising engine $h(y)$ when Γ is a first-order, scaled, 2D derivative operator (Eq. 1). This 2D formulation is a considerably more intricate problem to solve than the 1D case studied by Nielsen et al. [15], since the associated Fourier transform (Eq. 3) is not a ratio of polynomials in the Z-transform domain (and thus cannot be directly implemented as a single recursive filter). Furthermore, the frequency representation in Eq. 3 does not have a closed-form inverse (spatial representation).

Our idea is to decompose the 2D Tikhonov filter's kernel as a sum of axis-aligned 2D Gaussian functions, which we refer to as *Gaussian decomposition*. In this way, the filter can be approximated by a combination of Gaussian blurs, which can be implemented in $O(N)$ time with the recursive-filtering methods of Deriche [5] or Young and Vliet [19]. Gaussians are used as the building blocks of the decomposition due to their perfect separability (*i.e.*, $G(x,y) = G(x)G(y)$). This allows us to use symmetries of the Tikhonov filter to propose a novel and fast (< 1 ms) optimization scheme to compute the 2D Gaussian decomposition directly in the frequency domain, where the kernel has a closed-form expression (Eq. 3), and using 1D curve fitting. This optimization scheme is the main contribution of our work (Section III). We discuss the precision of the method and provide the execution times of our implementation (Section V), available at https://github.com/hermes-hf/Recursive-filtering-2D-Tikhonov-regularization .

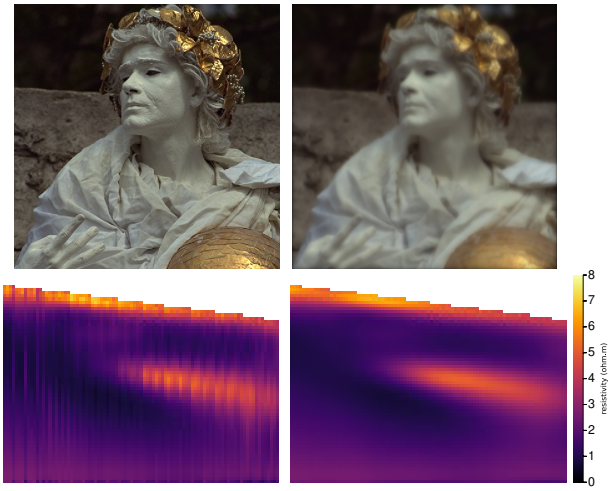


Fig. 1: Top left: original image. Top right: image filtered with the 2D Tikhonov kernel (parameters $\lambda_x = 10$ and $\lambda_y = 20$). Bottom left: geophysical inversion without regularization. Bottom right: inversion with Tikhonov regularization [2], [3].

II. Background and Related Works

Given an input g, the Tikhonov regularized solution f is a function that is close to g but is also smooth [15]:

$$E(f) = \frac{1}{2}\sum_{x,y}\left(f(x,y) - g(x,y)\right)^2 + \tag{1}$$
$$\lambda_x\Big(f(x,y) - f(x-1,y)\Big)^2 + \lambda_y\Big(f(x,y) - f(x,y-1)\Big)^2.$$

The functional $E(f)$ has smaller energy when f is close to g in the ℓ^2 norm and has a smooth derivative (this smoothness

is controlled by the λ_x and λ_y parameters). The minimum of this functional is the regularized solution f such that $\frac{d}{df(x,y)}E(f) = 0$. Differentiating Eq. 1 in $f(x,y)$ and equating to zero gives

$$f(x,y)\Big[1 + 2\lambda_x + 2\lambda_y\Big] - \lambda_x f(x-1,y) - \lambda_x f(x+1,y) - \lambda_y f(x,y+1) - \lambda_y f(x,y-1) = g(x,y). \quad (2)$$

Let the support of the input $g(x,y)$ be $x, y \in \{1, 2, ..., M\}$, then the total size of the input is $N = M^2$. f can be computed by solving the above linear system in the variables $f(x,y)$. Directly solving the system by Gaussian elimination has a complexity of $O(N^3)$. For periodic boundary conditions, a better solution is to use the Fast Fourier Transform (FFT) to compute the filter in the frequency domain, with complexity $O(N \log N)$: $\hat{f}(\omega,\xi) = \hat{K}(\omega,\xi)\,\hat{g}(\omega,\xi)$ where $\hat{\square}$ denotes the Discrete-time Fourier Transform (DTFT) of $\square$ and

$$\hat{K}(\omega,\xi) = \frac{1}{1 + 2\lambda_x(1-\cos(\omega)) + 2\lambda_y(1-\cos(\xi))}, \quad (3)$$

such implementation has been first considered by B. R. Hunt [11]. The regularized solution can be computed in $O(N)$ time with Multigrid methods [12], but these have higher overhead and memory requirements (Section V-A).

In the work of Nielsen et al. [15], 1D Tikhonov regularization is implemented with recursive filters in $O(N)$. However, their approach does not extend to the 2D case since the 2D kernel is not a ratio of polynomials in the Z domain (Eq. 3). Deriche and Abramatic [4] show that it is possible to approximate a 2D kernel as a sum of separable 1D recursive filters. Their work, however, is intended to be much more general, and so does not exploit the symmetries of the Tikhonov regularization. As a consequence, their method requires much more expensive SVD decomposition steps, with a computational complexity that scales in proportion to the Tikhonov filter's parameters (stronger regularization requires larger kernels and thus larger SVDs, taking hundreds to thousands of milliseconds). Our decomposition is computed in the frequency domain and its computation time is independent of the filter's parameters (λ_x and λ_y in Eq. 1), taking less than 1 millisecond to compute.

In a different approach, Farbman et al. [7] approximate given kernels by a sequence of convolutions with small separable kernels, in a multiscale scheme similar to a wavelet transform. Their method is fast and obtains $O(N)$ performance when filtering. However, finding the coefficients of the multiscale scheme for a given kernel requires a time-consuming non-linear optimization, which must be recomputed every time the filter's parameters change. Our Gaussian decomposition also must be recomputed for different filtering parameters, but it is practically instantaneous.

III. Our Gaussian Decomposition Method

Let $K(x,y)$ be the impulse response of the Tikhonov filter for a particular set of regularization parameters λ_x and λ_y, as illustrated in Fig. 2. $K(x,y)$ is given by the inverse DTFT

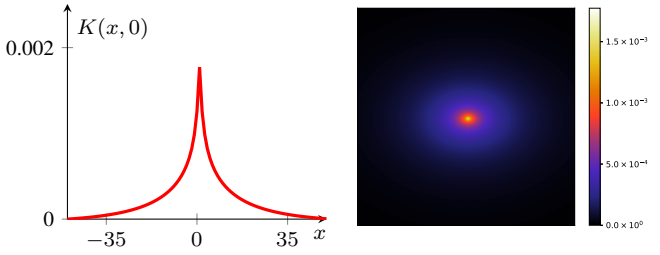


Fig. 2: $K(x,0)$ and heatmap of $K(x,y)$. $\lambda_x = 600$, $\lambda_y = 300$.

of $\hat{K}(\omega,\xi)$ (Eq. 3). To solve the regularization problem we propose to approximate the kernel as a sum of n_g Gaussian functions $G_s(x) = \frac{1}{\sqrt{2\pi s^2}} e^{-x^2/(2s^2)}$ in the form:

$$K(x,y) = \sum_{i=1}^{n_g} \alpha_i\, G_{\sigma_i}(x)\, G_{\gamma_i}(y). \quad (4)$$

Tikhonov filtering with K is then substituted by convolutions with Gaussians, which are efficiently computed by recursive filtering (Section IV). In other words, the regularization problem can be solved in $O(N)$, where N is the size of the input.

As discussed in the following sections, to compute the Gaussian decomposition we solve instead for

$$K(x,0) = \alpha_0\,\delta(x) + \sum_{i=1}^{n_g} \alpha_i\, G_{\sigma_i}(x), \quad (5)$$

where $\delta(x)$ is the unit impulse. We then use symmetries of the kernel (Section III-A) to obtain

$$K(x,y) = \alpha_0\,\delta(x,y) + \sum_{i=1}^{n_g} \alpha_i \sqrt{2\pi(\sigma_i\lambda_y/\lambda_x)^2}\, G_{\sigma_i}(x)\, G_{\sigma_i\lambda_y/\lambda_x}(y). \quad (6)$$

The unknown Gaussian amplitudes $\{\alpha_i\}$ and standard deviations $\{\sigma_i\}$ of this simplified decomposition can be obtained extremely fast with our Algorithm 1 (Section III-B), by successively fitting Gaussian terms to $\hat{K}(\omega,\xi)$ in the frequency domain (recall that the Fourier transform of a Gaussian is also a Gaussian). The amplitudes α_i can then be further improved by solving a small linear system (Section III-C).

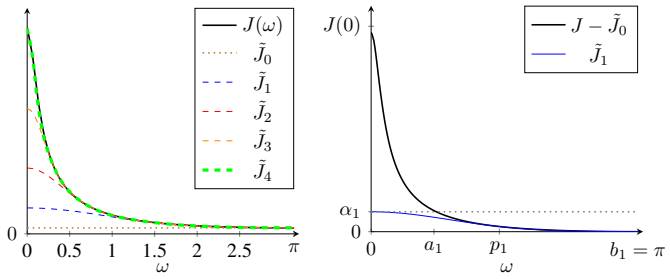
A. Elliptical Symmetry

We show that one can compute the kernel $K(x,0)$ at $y = 0$, and apply an elliptical rotation to approximate its exact value $K(x,y)$ at any given y value. The DTFT $\hat{K}$ (Eq. 3) can be written as a function of the variables l and θ:

$$\begin{cases} l(\omega,\xi) = \lambda_x \cos(\omega) + \lambda_y \cos(\xi), \\ \theta(\omega,\xi) = \tan^{-1}\left(\xi\,/\,\omega\right). \end{cases} \quad (7)$$

This is similar to a polar coordinate system. With this change of variables we have the following symmetry equation, which states the invariance of $\hat{K}$ with respect to θ:

$$\frac{d\hat{K}}{d\theta} = \frac{\partial\hat{K}}{\partial\omega}\frac{\partial\omega}{\partial\theta} + \frac{\partial\hat{K}}{\partial\xi}\frac{\partial\xi}{\partial\theta} = 0. \quad (8)$$



(a) (b)

Fig. 3: (a) Approximation in the frequency domain. Successively improving the solution by adding more Gaussian terms ($\lambda_x = 80$, $\lambda_y = 30$, and up to $n_g = 4$ Gaussians). (b) Fitting a Gaussian term $\alpha_0 + \alpha_1 \hat{G}_{\sigma_1}(\omega)$ such that the derivative at the intersection point p_1 coincides with $J'(p_1)$.

Replacing the appropriate expressions for $\frac{\partial \omega}{\partial \theta}$ and $\frac{\partial \xi}{\partial \theta}$ in Eq. 8, and then applying the inverse DTFT, one obtains the symmetry equation in the spatial domain (see Appendix C in the supplementary material for further details):

$$\lambda_x\, y \left(K(x+1,y) - K(x-1,y)\right) \\ -\lambda_y\, x \left(K(x,y+1) - K(x,y-1)\right) = 0.$$

Noticing that the central differences approximate the spatial derivatives of K: $2\,(\partial K/\partial x) \approx K(x+1,y) - K(x-1,y)$ and $2\,(\partial K/\partial y) \approx K(x,y+1) - K(x,y-1)$, one obtains:

$$\lambda_x\, y\, \frac{\partial K}{\partial x} - \lambda_y\, x\, \frac{\partial K}{\partial y} = 0.$$

This expression is in fact $\frac{d}{d\varphi} K(x,y) = 0$ in the elliptical coordinate system

$$\begin{cases} r(x,y) = \sqrt{x^2/\lambda_x \,+\, y^2/\lambda_y}, \\ \varphi(x,y) = \tan^{-1}\left(y \,/\, x\right). \end{cases}$$

With this, we are able to approximate $K(x,y)$ by rotating $K(x,0)$ in this coordinate system. That is:

$$K(x,y) \approx K\left(\sqrt{\lambda_x}\, r(x,y), 0\right).$$

Applying this rotation to the 1D Gaussian decomposition in Eq. 5 yields Eq. 6. Thus, with a Gaussian decomposition for only the x variable, one can obtain a decomposition for the whole x-y range.

B. Fast Gaussian Decomposition

In this section we describe a method to obtain a Gaussian decomposition for $K(x,0)$ very quickly. Our method consists of fitting Gaussian functions in the frequency domain.

The DTFT of $K(x,0)$ is given by

$$J(\omega) = 1 \Big/ \sqrt{(1 + 2\,\lambda_x + 2\,\lambda_y - 2\,\lambda_x \cos(w))^2 - 4\,\lambda_y^2}.$$

Thus, we are interested in solving

$$\alpha_0 + \sum_{i=1}^{n_g} \alpha_i \widehat{G}_{\sigma_i}(\omega) = J(\omega) \tag{9}$$

for $\omega \in [-\pi, \pi]$. Since all functions used are even, we only need to work with $\omega \in [0, \pi]$. Using Theorem 1 (Appendix A), explicit formulas for both sides of Eq. 9 are available (since the spatial domain is discrete, one must consider the aliasing effect in frequency space). We add a constant term α_0 to cover the case when the optimal standard deviation of a Gaussian term is close to zero, which would produce large numerical error (recall that the deviation in the frequency domain is the inverse of the deviation in the spatial domain for a Gaussian function). The term α_0 in the frequency domain corresponds to scaling the signal by α_0 in the spatial domain.

We propose a novel algorithm to find the α_i and σ_i values that solve Eq. 9, which is summarized in Algorithm 1. The function $J(\omega)$ is approximated for decreasing ω. The idea is to fit Gaussian terms starting with the largest (frequency-space) deviation to the smallest one (Fig. 3a). We expect that the Gaussians with smaller deviation should not interfere with the terms with larger deviation, due to the rapid decay in ω for Gaussian functions. We first fit the term of deviation ∞, which means defining the constant amplitude term α_0. It is set as $\alpha_0 = J(\pi)$, which is the smallest value attained by $J(\omega)$.

Next, we proceed by computing initial values for the amplitudes $\alpha_1, \ldots, \alpha_{n_g}$. Experimentally, we observed that uniformly spaced values work well. Thus, we select amplitudes such that $\alpha_i = i\,\alpha_1$ (for $i \geq 1$) and $\alpha_0 + \cdots + \alpha_{n_g g} = J(0)$.

The next step is to find the deviations $\sigma_1, \ldots, \sigma_{n_g}$. This is done in an iterative fashion: we find the best σ_1 for the partial fit $\tilde{J}_1(\omega, \sigma_1) = \alpha_0 + \alpha_1 \hat{G}_{\sigma_1}(\omega)$, then we fix σ_1 and find the best σ_2 for the partial fit $\tilde{J}_2(\omega, \sigma_2) = \alpha_0 + \alpha_1 \hat{G}_{\sigma_1}(\omega) + \alpha_2 \hat{G}_{\sigma_2}(\omega)$, and so on up to σ_{n_g}.

At the j-th step, the best deviation σ_j will be the one which makes the partial fit $\tilde{J}_j(\omega, \sigma_j)$ match $J(\omega)$ in value and in derivative at some point p_j, that is, $\tilde{J}_j(p_j, \sigma_j) = J(p_j)$ and $\tilde{J}'_j(p_j, \sigma_j) = J'(p_j)$. The point $p_j \in [0, \pi]$ is the frequency location where the graph of $\tilde{J}_j$ will be tangent to J (see Fig. 3b for an illustration of p_1 and $\tilde{J}_1$). Observe that any intersection point p (where simply $\tilde{J}_j(p, \sigma) = J(p)$) will uniquely determine σ, since there will be only one possible choice of σ which will make $\tilde{J}_j(p, \sigma) = J(p)$ at p (this occurs because the Gaussian $\hat{G}_\sigma(\omega)$ is monotonous decreasing in σ). In other words, for a fixed intersection point p, the associated σ is given implicitly as the solution to $\tilde{J}_j(p, \sigma) = J(p)$. Thus, to find σ_j, we proceed by selecting a candidate point q, computing the value of σ^q associated with q by solving $\tilde{J}_j(q, \sigma^q) = J(q)$, and then checking whether the derivatives' matching error $D_j(q) = \tilde{J}'_j(q, \sigma^q) - J'(q)$ is within some error tolerance: $|D_j(q)| < \varepsilon_1$. If yes, then we found the j-th deviation $\sigma_j = \sigma^q$ with associated $p_j = q$. Otherwise, this candidate is rejected and we select a new candidate point q to repeat the process. In our implementation $\varepsilon_1 = 10^{-7}$.

To avoid a brute-force search of candidate points across the whole domain $[0, \pi]$, we use a binary search scheme. For the j-th step, the first candidate point q will be the midpoint of the interval $[a_j, b_j]$ (described below). If this candidate is rejected (because $|D_j(q)| \geq \varepsilon_1$), and if $D_j(q)$ is negative, we repeat the binary search with the lower half of the interval (*i.e.*, $[a_j, q]$).

If $D_j(q)$ is positive, we select the upper half (*i.e.*, $[q, b_j]$). This procedure is repeated until a candidate point is accepted withing the error tolerance, or for a maximum of T binary search iterations (in our implementation, $T = 10$).

Extra care must be taken when choosing the upper interval, as we expect the new Gaussian term to interfere more with the previously fitted terms. Thus, to avoid any significant interference, we only choose the upper interval if $J(q) - \tilde{J}_{j-1}(q)$ is larger than some threshold ε_2 ($5 \cdot 10^{-4}$ in our implementation), which means there is still room to fit a Gaussian term without exceeding the value of $J(\omega)$ at $\omega = q$. Finally, for the initial search interval $[a_j, b_j]$, we choose a_j to be the point where the constant function α_j (the largest value the j-th Gaussian term can attain) intersects $J(q) - \tilde{J}_{j-1}(q)$ (the residual of the fit from the previous, $(j-1)$-th step). Furthermore, the upper limit $b_j = p_{j-1}$ is initialized as the intersection point chosen in the previous step, and $b_1 = \pi$ (Fig. 3b).

Our method works best when $\lambda_x \geq \lambda_y$, if this is not the case, we swap those values before applying the algorithm and swap them back after running the algorithm. This is equivalent to computing the Gaussian decomposition for $K(0, y)$ instead of $K(x, 0)$ (followed by the appropriate elliptical rotation).

C. Optimizing the Amplitudes

After solving for the deviations, the Gaussian amplitudes can be further improved by solving a 1D linear least squares problem in Eq. 9. We fix the previously obtained $\{\sigma_i\}$ values and solve for the amplitude variables $\{\alpha_i\}$. This is done by discretizing the equation using 100 linearly spaced ω values in $[0, \pi]$. Furthermore, we used a large weight (10^3) associated with the linear equation at $\omega = 0$, to enforce a proper representation of the zero frequency value $J(0)$ (thus preserving the area under the Tikhonov kernel). The resulting optimization matrix might be ill-conditioned when two or more σ values are similar. To get around this, we used the `lsmr` iterative solver from Julia's `IterativeSolvers.jl` package. This gives a very lightweight optimization scheme with complexity $O(n_g)$ on the number of Gaussians.

IV. Implementation Details

We implemented Algorithm 1 in the Julia programming language using the parameters $T = 10$ (max iterations per Gaussian), $\varepsilon_1 = 10^{-7}$ (error tolerance in the derivative of $\tilde{J}$) and $\varepsilon_2 = 5 \cdot 10^{-4}$ (error tolerance in the value of $\tilde{J}$). The function inverses $F^{-1}(\alpha_i)$ and $H^{-1}(F(p))$ can be computed through a binary search since both F and H are decreasing. Note that the running time of Algorithm 1 is independent of the filtered signal size. Its complexity is $O(n_g)$ where n_g is the number of Gaussian functions in the decomposition, usually $n_g \in \{4, 5, 6\}$. Since we have explicit formulas for G and F, the derivatives can be computed with automatic differentiation.

Given a Tikhonov kernel $K(x, y)$ with parameters λ_x and λ_y, we compute its Gaussian decomposition using Algorithm 1 and Eq. 6. We then compute the convolution of the input signal with each Gaussian by recursive filtering, followed by summing the results, to generate the final output. We perform

Algorithm 1: Gaussian decomposition of $J(\omega)$

input: DTFT $J(\omega)$, number of Gaussian terms $n_g \in \mathbb{N}$
output: $(\alpha_1, ..., \alpha_{n_g}), (\sigma_1, ..., \sigma_{n_g}) \in \mathbb{R}^{n_g}$ and $\alpha_0 \geq 0$
$p_0 \leftarrow \pi$; $\alpha_0 \leftarrow J(\pi)$
for $j = 1, 2..., n_g$ **do**
  $\alpha_j \leftarrow (J(0) - \alpha_0)\ j\ /(\sum_{k=1}^{n_g} k)$
  $F(\omega) := J(\omega) - \alpha_0 - \sum_{k=1}^{j-1} \alpha_k \hat{G}_{\sigma_k}(\omega)$ //$J - \tilde{J}_{j-1}$
  $a_j \leftarrow F^{-1}(\alpha_j)$; $b_j \leftarrow p_{j-1}$
  for $t = 1, 2, ..., T$ **do**
    $q \leftarrow (a_j + b_j)/2$
    $H(s) := \hat{G}_s(q)$
    $\sigma \leftarrow H^{-1}(F(q))$
    $D_j = \alpha_j \hat{G}'_\sigma(q) - F'(q)$ //$\tilde{J}'_j(q, \sigma) - J'(q)$
    if $D_j < -\varepsilon_1$ **then**
      $b_j \leftarrow q$
    else if $D_j > \varepsilon_1$ **and** $F(q) \geq \varepsilon_2$ **then**
      $a_j \leftarrow q$
    else
      break
    end if
  end for
  $\sigma_j \leftarrow \sigma$; $p_j \leftarrow q$
end for
Optimize $(\alpha_1, \ldots, \alpha_{n_g})$ by linear solve (Section III-C)

recursive filtering using Deriche's method [5]. Another option would be the method of Young and Vliet [19], but it is less precise when the deviation of a Gaussian term is small, which occurs often when approximating the peak of the Tikhonov kernel K. We wrote a Julia implementation of the Deriche filter which was used for interactive analysis, and a C++ implementation which was optimized to reduce execution time.

The Z-Transform of Deriche's 1D Gaussian filter is

$$\begin{aligned}(\eta_3 z^{-3} + \eta_2 z^{-2} + \eta_1 z^{-1} + \eta_0 + \eta_1 z + \eta_2 z^2 + \eta_3 z^3) \\ \cdot\ (1 + d_1 z + d_2 z^2 + d_3 z^3 + d_4 z^4)^{-1} \\ \cdot\ (1 + d_1 z^{-1} + d_2 z^{-2} + d_3 z^{-3} + d_4 z^{-4})^{-1}\,;\end{aligned} \tag{10}$$

where the values of the constants depend on the Gaussian's σ and are displayed in Appendix B (supplementary material). To perform 2D Gaussian filtering we need to filter both columns and rows in sequence. For the column filter we first apply the block filter $\text{block}(y) = \sum_{i=-3}^{3} \eta_i \,\text{in}(y + i)$, followed by the causal (forward) filter

$$\text{fwd}(y) = \text{block}(y) - \sum_{i=1}^{4} d_i \,\text{fwd}(y - i)\,,$$

and finally the anticausal (backward) filter

$$\text{out}(y) = \text{bwd}(y) = \text{fwd}(y) - \sum_{i=1}^{4} d_i \,\text{bwd}(y + i)\,.$$

For the row filter, we choose to apply the block filter after the forward and backward filters, as this allows us to use a single

Fig. 4: Comparison between the FFT filtered image (left) and our method (right), with $\lambda_x = 100$ and $\lambda_y = 40$.

buffer for the whole 2D filter. Note that the column filter and the row filter will generally have different σ parameters (the Gaussians are axis-aligned but not necessarily isotropic). As originally described by Deriche [5], the Gaussian filter could be implemented with forward and backward passes in parallel (out = fwd + bwd). However, we chose an implementation in series (out = bwd ∘ fwd) for better performance, as it requires no separate memory buffers for both passes, and thus makes a better use of the CPU's cache. The boundary conditions for the filter can be selected according to each application, in a variety of ways as described by Nehab and Maximo [14]. We use zero extension in our experiments.

V. Results

Our method obtains good numerical accuracy. We use the FFT-filtered result as the ground truth for our comparison (Fig. 4). As a note, using the FFT to filter a signal implies a periodic boundary condition, which is commonly not adequate for most Tikhonov regularized problems. To compute the ground truth results, we pad the images with zeros before executing the FFT to minimize the effect of the periodicity.

Table I displays the numerical accuracy of the resulting Tikhonov filter using our Gaussian decomposition. Note that increasing the number of Gaussians from $n_g = 5$ to $n_g = 6$ does not improve accuracy, since our Algorithm 1 is not guaranteed to find a global optimum.

If more precision is desired (at the cost of computation time), it is possible to apply a non-linear least squares optimizer in the spatial domain to simultaneously solve for amplitudes and deviations of the Gaussians. In this case, the model function (with optimization variables A_i, B_i and C_i) is

$$K(x,y) = A_0^2\,\delta(x,y) + \sum_{i=1}^{n_g} A_i^2\, e^{-\frac{x^2}{2}C_i^2 - \frac{y^2}{2}B_i^2}\,.$$

This function is used since it is best to use unnormalized Gaussian terms and squared variables so that the problem becomes convex (and easier to optimize). In practice, instead of optimizing with respect to $K(x,y)$, we apply the optimization with $K(x,0)$ and $K(0,y)$, reducing the dimensionality of the problem. The function $K(x,0)$ can be computed using the IFFT on $J(\omega)$, $K(0,y)$ can be computed similarly. The IFFTs are computed in $O(1)$ since we always use a fixed number of 200 samples, and the non-linear optimizer runs in $O(n_g)$ since we use a fixed number of 20 iterations in the non-linear solver. As initial conditions for the solver we use the outputs of Algorithm 1 (but without the linear optimization on α_i).

Metric	$n_g = 4$	$n_g = 5$	$n_g = 6$
↓ avg. ℓ^1	$0.99\% \pm 0.35\%$	$0.50\% \pm 0.32\%$	$1.23\% \pm 0.40\%$
↓ max ℓ^1	2.30%	1.45%	1.74%
↓ avg. ℓ^2	$1.05\% \pm 0.34\%$	$0.57\% \pm 0.33\%$	$1.27\% \pm 0.38\%$
↓ max ℓ^2	2.32%	1.64%	1.76%
↑ avg. PSNR	$47.1\text{dB} \pm 3.1\text{dB}$	$53.2\text{dB} \pm 4.1\text{dB}$	$45.6\text{dB} \pm 3.4\text{dB}$
↑ min PSNR	38.1dB	42.0dB	40.6dB

TABLE I: Average error and PSNR over 10 images from the Kodak dataset, over all pairs of λ_x and λ_y in $\{10, 20, ..., 100\}$. FFT-filtered result used as ground truth for comparison, while varying the number n_g of Gaussian terms used in the approximation. Smaller ↓ ℓ^1/ℓ^2 and larger ↑ PSNR is better.

This non-linear optimization only shows improvements with $n_g \geq 6$ Gaussian terms. For $n_g = 6$, relative errors decrease to $0.24\% \pm 0.12\%$ (avg. ℓ^1) and $0.27\% \pm 0.12\%$ (avg. ℓ^2), and the average PSNR increases to $59.3\text{dB} \pm 3.8\text{dB}$. This non-linear optimization process takes 110 milliseconds to compute, compared to less than 0.4 ms for our Algorithm 1.

When both λ_x and λ_y are smaller than 5, the accuracy of the Gaussian decomposition decreases, even when using the non-linear optimization (around 33dB PSNR). In this case, the kernel $K(x,y)$ is not sufficiently elliptically symmetrical. However, small values of λ are not very useful as the regularization will be nearly non-existent.

A. Computational Performance

We compare the filtering execution time of our method against the FFT, implemented with the C++ FFTW library. This library is highly optimized and extremely fast, since it contains several hard-coded transforms of small sizes, even though the FFT is $O(N \log N)$. Our recursive fultering method is $O(N)$ and implemented in C++. Multigrid methods also run in $O(N)$, however, they are slower in practice than their complexity suggests, and also require larger amounts of memory. We use the C++ Multigrid implementation of Kazhdan and Hoppe [12] in our experiments. All tests were done with a single thread on an Intel i5 2.90 GHz processor.

Fig. 5 summarizes the execution times. The FFTW library has different "planning" options: Estimate (fastest), Measure (slower), and Patient (very slow). Each option has a trade-off between planning time and the FFT execution time. The planning time is how much it takes for the FFTW to determine the optimal sequence of steps to apply the FFT for the particular CPU where the code is being executed. The advantage is that plans can be reused for inputs of the same size. In Fig. 5 we display the execution times with and without the planning times for the FFTW.

Our implementation is faster than the Estimate option for images larger than 2048×2048 and always faster than the Measure option when accounting for the planning time (which takes more than 500 ms even for small 1024×1024 inputs). If the planning time is not included, our implementation is faster than the Measure option for inputs of size larger than or equal

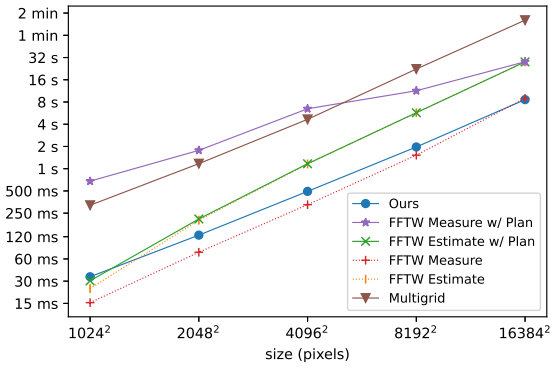


Fig. 5: Filtering execution time (lower is better). Comparison between our method with 5 Gaussian terms using the Deriche filter, different FFTW options, and Multigrid. We sampled 10 executions of each algorithm for each size of image and chose the minimum time. Note the $O(N)$ scaling of our method vs $O(N \log N)$ for the FFT in this log–log plot.

to 16384×16384. Note that the state-of-the-art FFTW library has been very carefully tuned to generate optimal code for each machine. Our particular implementation of the Deriche filter is intended as a proof of concept. With more work, faster implementations can be achieved. We also note that the FFT is limited to periodic boundary conditions, and other types of input extension must be performed by padding the signal, thus increasing memory and computation time (especially for strong regularizations which require large values of λ_x and λ_y, resulting in larger kernels and larger padding requirements). Finally, our method is one order of magnitude faster than Multigrid.

VI. Conclusion and Future Work

We provided an insight into the symmetries of the 2D Tikhonov kernel, and how we can exploit them to obtain a Gaussian decomposition. We described a novel algorithm to compute this decomposition extremely fast and discussed the precision of the method. We also compared the execution times of our implementation with other methods, mainly the FFTW.

We obtained a precise method by only using 1D information of the 2D kernel. The complete regularization problem can then be solved in $O(N)$, faster than existing Multigrid methods and faster than the FFT for large inputs. Although we only used a single-threaded CPU implementation, the recursive filter has many steps which can be executed in parallel (in particular, each row/column can be filtered independently). Performance can also be improved by using GPU implementations of recursive filters as proposed by Nehab et al. [13].

Our method can be used jointly with edge-aware filters such as the first-order recursive implementation of the Gaussian filter provided by Gastal and Oliveira [8]. This is useful when it is desirable to use non-homogeneous weights for the filter. It is then possible to avoid excessive smoothing along sharp edges on the data while retaining the $O(N)$ complexity. This is not possible with the FFT.

For future work we plan to investigate how these ideas can be adapted to higher-dimensional Tikhonov regularization, and plan to give an analytical description of how non-homogeneous regularization relates to our implementation. We also plan to obtain an efficient solution for the continuous version of the Tikhonov regularization filter with our method.

Acknowledgements. This work was partially supported by CNPq-Brazil (436932/2018-0), Petrobras (2017/00752-3), FAPERGS (22/2551-0000619-1) and financed in part by "Coordenação de Aperfeiçoamento de Pessoal de Nível Superior" - Brasil (CAPES) - Finance Code 001

References

[1] A. Bouhamidi and K. Jbilou, "Sylvester tikhonov-regularization methods in image restoration," *Journal of Computational and Applied Mathematics*, vol. 206, no. 1, pp. 86–98, 2007.
[2] B. R. Carvalho and P. T. L. Menezes, "Marlim R3D: a realistic model for CSEM simulations-phase I: model building," *Brazilian Journal of Geology*, vol. 47, pp. 633–644, 2017.
[3] J. L. Correa and P. T. L. Menezes, "Marlim R3D: A realistic model for controlled-source electromagnetic simulations-Phase 2: The controlled-source electromagnetic data set," *Geophysics*, vol. 84, no. 5, 2019.
[4] R. Deriche and J. Abramatic, "Design of 2-D recursive filters using singular value decomposition techniques," in *IEEE International Conf. on Acoustics, Speech, and Signal Processing*, pp. 03–05.
[5] R. Deriche, "Recursively implementing the gaussian and its derivatives," in *Proc. Secound Int. Conf. On Image Processing*, 1992, pp. 263–267.
[6] H. Egger and H. Engl, "Tikhonov regularization applied to the inverse problem of option pricing: Convergence analysis and rates," *Inverse Problems*, vol. 21, 2005.
[7] Z. Farbman, R. Fattal, and D. Lischinski, "Convolution pyramids," *ACM Trans. Graph.*, vol. 30, p. 175, 12 2011.
[8] E. S. L. Gastal and M. M. Oliveira, "High-order recursive filtering of non-uniformly sampled signals for image and video processing," *Computer Graphics Forum*, vol. 34, no. 2, 2015.
[9] A. Gholami and H. R. Siahkoohi, "Regularization of linear and non-linear geophysical ill-posed problems with joint sparsity constraints," *Geophysical Journal International*, vol. 180, no. 2, pp. 871–882, 2010.
[10] A. E. Hoerl and R. W. Kennard, "Ridge regression: Biased estimation for nonorthogonal problems," *Technometrics*, vol. 12, no. 1, 1970.
[11] B. R. Hunt, "The application of constrained least squares estimation to image restoration by digital computer," *IEEE Transactions on Computers*, vol. C-22, no. 9, pp. 805–812, 1973.
[12] M. Kazhdan and H. Hoppe, "Streaming Multigrid for Gradient-Domain Operations on Large images," *ACM Trans. Graphics*, vol. 27(3), 2008.
[13] D. Nehab, A. Maximo, R. Lima, and H. Hoppe, "GPU-efficient recursive filtering and summed-area tables," *ACM TOG*, vol. 30, no. 6, 2011.
[14] D. Nehab and A. Maximo, "Parallel recursive filtering of infinite input extensions," *ACM Trans. Graph.*, vol. 35, no. 6, 2016.
[15] M. Nielsen, L. Florack, and R. Deriche, "Regularization, scale space, and edge detectors," *Journ. of Math. Imaging and Vision*, vol. 7, 1997.
[16] Y. Romano, M. Elad, and P. Milanfar, "The little engine that could: Regularization by denoising (red)," *SIAM J. on Imaging Sciences*, 2016.
[17] A. N. Tikhonov and V. Y. Arsenin, *Solutions of Ill-posed problems*. W.H. Winston, 1977.
[18] M. Vauhkonen, D. Vadász, P. Karjalainen, E. Somersalo, and J. Kaipio, "Tikhonov regularization and prior information in electrical impedance tomography," *IEEE Trans. on Medical Imaging*, vol. 17, 1998.
[19] I. Young and L. Van Vliet, "Recursive implementation of the gaussian filter," *Signal Processing*, vol. 44, pp. 139–151, 06 1995.

Appendix A

Theorem 1. *The DTFT of a Gaussian G_σ is approximately $\hat{G}_\sigma(\omega) \approx \sum_{k=-\tau}^{\tau} \exp(-\frac{(\omega-2\pi k)^2}{2}\sigma^2)$. For $\sigma \geq 0.4$, using $\tau = 2$ gives an ℓ^∞ error of $5.68 \cdot 10^{-9}$. Gaussians with $\sigma < 0.4$ in space can be treated as constant functions in frequency.*

See the supplementary material for a detailed proof.