Adaptive double-phase ROF model: report on 1D experiments

Wojciech Górny* Michał Łasica† Alexandros Matsoukas‡

September 30, 2025

Abstract

This note reports the results of several numerical experiments involving the adaptive double-phase ROF model in the one-dimensional case.

Contents

*Faculty of Mathematics, Universität Wien, Oskar-Morgerstern-Platz 1, 1090 Vienna, Austria; Faculty of Mathematics, Informatics and Mechanics, University of Warsaw, Banacha 2, 02-097 Warsaw, Poland; wojciech.gorny@univie.ac.at

†Institute of Mathematics of the Polish Academy of Sciences, Śniadeckich 8, 00-656 Warsaw, Poland; mlasica@impan.pl

‡Department of Mathematics, School of Applied Mathematical and Physical Sciences, National Technical University of Athens, Zografou Campus, 157 80 Athens, Greece; alexmatsoukas@mail.ntua.gr

1 Introduction

Below, we briefly describe some numerical experiments concerning the following model in image processing. Denote by $g \in L^2(\Omega)$ a noisy image. In order to reduce the staircasing effect of the classical ROF model [4]

$$\min \left\{ \mathrm{TV}(v) + \frac{1}{2\lambda} \int_\Omega |v-g|^2 \,\mathrm{d}x : \ v \in L^2(\Omega) \right\} \tag{1}$$

we propose a new approach called the *adaptive double-phase ROF model* which is briefly described below.

Algorithm 1 The adaptive double-phase ROF model

1: Fix a noisy image g and choose a parameter $\lambda > 0$ suitable for the classical ROF minimization problem (1).
2: Find a solution u_{ROF} of the classical ROF minimization problem (1) for the datum g and the parameter λ.
3: Choose a mollification radius $\rho > 0$ and a non-increasing function W with compact support and $W(0) > 0$.
4: Mollify the function u_{ROF} with a kernel of radius ρ; denote the result by $\tilde{u}_{\mathrm{ROF}}$.
5: Compute the weight w using the formula

$$w(x) = W(|\nabla \tilde{u}_{\mathrm{ROF}}|(x))$$

from the mollified function $\tilde{u}_{\mathrm{ROF}}$.
6: Find a solution u_{dpROF} of the double-phase ROF minimization problem

$$\min \left\{ \mathrm{TV}(v) + \int_\Omega w(x)|\nabla v|^2 \,\mathrm{d}x + \frac{1}{2\lambda} \int_\Omega |v-g|^2 \,\mathrm{d}x : \ v \in L^2(\Omega) \right\}. \tag{2}$$

for the datum g, the parameter λ and the weight w.

Well-posedness of problem (2) was considered e.g. in [3, 2]. The main idea behind the algorithm above is to use the classical ROF model to detect edges in order to fine-tune the regularization in the next step, so that heuristically the image is processed by the 1-Laplacian operator close to the edges in order to ensure that they are well-preserved, while far from the jumps of the original image we add a Laplacian term with a large weight to reduce staircasing.

The above model contains several free parameters whose effect we present in this note: the coefficient λ, which already appears in the classical ROF model (1) and in both models is used to balance the regularization and fidelity terms; the mollification radius ρ (very small, in the implementation: 2-5 pixels); and the weight W. It needs to vanish at some small number $r > 0$ so that support of the resulting weight w is away from the (sufficiently large) jumps of u_{ROF}. The value $W(0)$ and the shape of the function between 0 and r determine whether the regularization effects of Laplacian or the 1-Laplacian is stronger away from the jumps of u_{ROF}. For modeling purposes, we consider three different types of weights. They are subsequently denoted as:

Weight 1:

$$W_1(x) = \max\left(0, a - b\max\left(x, \frac{a}{2b}\right)\right);$$

Weight 2:

$$W_2(x) = \max(0, a - bx);$$

Weight 3:

$$W_3(x) = h\chi_{[0,r]}.$$

Note that all the weights have the desired behavior, i.e., they are non-increasing and compactly supported. The first one is first constant and then linearly decreasing; the second one is linearly decreasing; and the third one is piecewise constant with a single jump.

The numerical experiments reported here were done using an implementation of the adaptive double-phase ROF model through the Chambolle–Pock algorithm [1]. Thanks to the uniform convexity of the L^2-distance term we may use the accelerated version of this algorithm. The algorithm was implemented in Python using the Google Colab platform. Clearly, the size of the grid cannot be too small, since if we rescale a given picture to a smaller resolution the gradients become large and the effect of the added weights becomes smaller (and ultimately they are equal to zero, at which point we have the usual ROF model). In the one-dimensional experiments presented below, the grid size is in the range 343-4032 pixels (usually 1000).

The aim of this note is to provide a detailed comparison of the behavior of results of the Chambolle–Pock algorithm solving the classical ROF problem versus the double-phase ROF problem with adaptive weights of type W_1, W_2, W_3 and the Huber ROF model

$$\min\left\{\int_\Omega |\nabla v|_\alpha \,\mathrm{d}x + \frac{1}{2\lambda}\int_\Omega |v-f|^2\,\mathrm{d}x\colon\ v \in L^2(\Omega)\right\},$$

where

$$|\xi|_\alpha = \begin{cases} \frac{1}{2\alpha}|\xi|^2 & \text{if } |\xi| \le \alpha; \\ |\xi| - \frac{\alpha}{2} & \text{if } |\xi| > \alpha. \end{cases}$$

The first two comparisons are of technical nature: we compare the performance of the standard (non-accelerated) Chambolle–Pock algorithm and the accelerated Chambolle–Pock algorithm. Since the results of the two algorithms are very close in the tested measures, we subsequently use the accelerated algorithm. Then, we compare the performance of these algorithms at different levels of tolerance (i.e., the value of the stop parameter in the Chambolle–Pock algorithm). Subsequently, we use the tolerance levels of 10^{-4} and 10^{-6} for further experiments. We then show the main results in terms of optimization of the performance of the algorithm. We study the effect of the free parameters, namely: the radius of mollification of the ROF-minimizer; the values of λ; the weights; and performance over different noise levels. We test the algorithms over a set of synthetic and natural images (or more precisely, single rows of natural images).

2 Comparison of the accelerated/non-accelerated algorithms

We first report several comparisons between the results of the classical ROF minimization problem, the ROF problem with adaptive weights, the Huber ROF problem, and the accelerated versions of each of the algorithms. The corresponding file with the python code is named '1D-acceleration'. The purpose of this is the verification of the accuracy of the accelerated and non-accelerated algorithms and their comparison at different tolerance levels in the Chambolle–Pock algorithm. For this experiment, we choose the following parameters:

1. $\lambda = 0.24$;
2. $\sigma = 0.01$;
3. Tolerance level: 10^{-4} and 10^{-6};
4. Radius of mollification: 2;
5. Weight 1: $a = 10^3$, $b = 10^4$;
6. Weight 2: $a = 10^3$, $b = 10^4$;
7. Weight 3: $h = 10^3$, $r = 0.03$.

We present the results for two test functions (images):

1. A characteristic function of an interval, denoted
$$\text{jump} = \chi_{[\frac{1}{4},\frac{3}{4}]};$$
2. A function with several jumps over different scales and linear interpolation between them, denoted
$$\text{saw}(x) = x\chi_{[0,\frac{1}{4}]} + 2(x - \frac{1}{4})\chi_{[\frac{1}{4},\frac{1}{2}]} + 3(x - \frac{1}{2})\chi_{[\frac{1}{2},\frac{3}{4}]} + 4(x - \frac{3}{4})\chi_{[\frac{3}{4},1]}.$$

In all the 1D experiments, we track the following several measures of performance of the considered algorithms. Below, we denote by "image" the original image, by "noisy" the original image subject to (Gaussian) noise, and by "result" the result of the reconstruction.

1. The normalized total variation distance to the original image, i.e.,
$$\text{d}_{\text{TV}}^{\text{image}}(\text{result}) = \frac{\text{TV}(\text{result} - \text{image})}{\text{TV}(\text{image})};$$
2. The normalized L^2 distance to the original image, i.e.,
$$\text{d}_{L^2}^{\text{image}}(\text{result}) = \frac{\|\text{result} - \text{image}\|_2}{\|\text{image}\|_2};$$

3. The normalized L^2 distance to the noisy image, i.e.,

$$\mathrm{d}_{L^2}^{\mathrm{noisy}}(\mathrm{result}) = \frac{\|\mathrm{result} - \mathrm{noisy}\|_2}{\|\mathrm{image}\|_2};$$

4. The number of iterations required to complete the algorithm.

Here and in this entire note, with the exception of the last section containing pictures, the values of these measures are averaged over 100 runs of the algorithm: in each run, we generate a noisy image by adding Gaussian noise of a prescribed level and run all the algorithms on the same newly generated image. We present the results only for the above specific choice of parameters and test functions, but similar behavior is reproduced for other choices as well.

Table 1: err $= 10^{-4}$, $f =$ jump

Standard	Classical ROF	Weight 1	Weight 2	Weight 3	Huber ROF
$\mathrm{d}_{\mathrm{TV}}^{\mathrm{image}}$	0.45	0.39	0.39	0.41	0.54
$\mathrm{d}_{L^2}^{\mathrm{image}}$	0.060	0.060	0.060	0.060	0.060
$\mathrm{d}_{L^2}^{\mathrm{noisy}}$	0.080	0.081	0.081	0.081	0.079
Iterations	1155	1242	1250	1243	989
Accelerated	Classical ROF	Weight 1	Weight 2	Weight 3	Huber ROF
$\mathrm{d}_{\mathrm{TV}}^{\mathrm{image}}$	0.36	0.23	0.22	0.24	0.47
$\mathrm{d}_{L^2}^{\mathrm{image}}$	0.059	0.059	0.059	0.060	0.060
$\mathrm{d}_{L^2}^{\mathrm{noisy}}$	0.080	0.081	0.081	0.081	0.079
Iterations	708	525	549	551	242

Table 2: err $= 10^{-4}$, $f =$ saw

Standard	Classical ROF	Weight 1	Weight 2	Weight 3	Huber ROF
$\mathrm{d}_{\mathrm{TV}}^{\mathrm{image}}$	1.18	0.43	0.40	0.47	1.09
$\mathrm{d}_{L^2}^{\mathrm{image}}$	0.071	0.065	0.066	0.063	0.070
$\mathrm{d}_{L^2}^{\mathrm{noisy}}$	0.216	0.230	0.231	0.227	0.216
Iterations	1266	1750	1995	1978	1169
Accelerated	Classical ROF	Weight 1	Weight 2	Weight 3	Huber ROF
$\mathrm{d}_{\mathrm{TV}}^{\mathrm{image}}$	1.26	0.48	0.44	0.55	1.16
$\mathrm{d}_{L^2}^{\mathrm{image}}$	0.072	0.070	0.075	0.076	0.071
$\mathrm{d}_{L^2}^{\mathrm{noisy}}$	0.215	0.229	0.232	0.229	0.215
Iterations	613	426	525	543	259

Table 3: err $= 10^{-6}$, $f =$ jump

Standard	Classical ROF	Weight 1	Weight 2	Weight 3	Huber ROF
$\mathrm{d}_{\mathrm{TV}}^{\mathrm{image}}$	0.34	0.23	0.22	0.23	0.45
$\mathrm{d}_{L^2}^{\mathrm{image}}$	0.060	0.059	0.059	0.059	0.060
$\mathrm{d}_{L^2}^{\mathrm{noisy}}$	0.080	0.082	0.082	0.082	0.080
Iterations	15506	21402	24099	24128	7583
Accelerated	Classical ROF	Weight 1	Weight 2	Weight 3	Huber ROF
$\mathrm{d}_{\mathrm{TV}}^{\mathrm{image}}$	0.34	0.22	0.21	0.23	0.45
$\mathrm{d}_{L^2}^{\mathrm{image}}$	0.060	0.059	0.059	0.059	0.060
$\mathrm{d}_{L^2}^{\mathrm{noisy}}$	0.080	0.082	0.082	0.082	0.080
Iterations	7175	2852	2755	2778	2015

Table 4: err $= 10^{-6}$, $f =$ saw

Standard	Classical ROF	Weight 1	Weight 2	Weight 3	Huber ROF
$\mathrm{d}_{\mathrm{TV}}^{\mathrm{image}}$	1.26	0.49	0.44	0.57	1.16
$\mathrm{d}_{L^2}^{\mathrm{image}}$	0.073	0.072	0.077	0.080	0.072
$\mathrm{d}_{L^2}^{\mathrm{noisy}}$	0.216	0.230	0.233	0.230	0.216
Iterations	7118	14636	18599	19234	5685
Accelerated	Classical ROF	Weight 1	Weight 2	Weight 3	Huber ROF
$\mathrm{d}_{\mathrm{TV}}^{\mathrm{image}}$	1.26	0.49	0.44	0.57	1.16
$\mathrm{d}_{L^2}^{\mathrm{image}}$	0.073	0.072	0.077	0.079	0.072
$\mathrm{d}_{L^2}^{\mathrm{noisy}}$	0.216	0.230	0.233	0.230	0.216
Iterations	4166	3633	4622	4616	2416

We also study the difference between the result of the accelerated algorithm vs the result of the non-accelerated algorithm in the L^2 norm and in the total variation. More precisely, we consider the following two additional measures. Below, we denote by "image" the original image, by "std" the result of the reconstruction with the non-accelerated algorithm, and by "acc" the result of the reconstruction with the accelerated algorithm.

1. The normalized total variation distance between the accelerated and non-accelerated algorithm, i.e.,
$$\mathrm{d}_{\mathrm{TV}}^{\mathrm{acc}} = \frac{\mathrm{TV}(\mathrm{acc} - \mathrm{std})}{\mathrm{TV}(\mathrm{image})};$$
2. The normalized L^2 distance to the original image, i.e.,
$$\mathrm{d}_{L^2}^{\mathrm{acc}} = \frac{\|\mathrm{acc} - \mathrm{std}\|_2}{\|\mathrm{image}\|_2}.$$

We present the results below in Tables 5-6.

Table 5: $f = \text{jump}$

$\mathrm{d}_{\mathrm{TV}}^{\mathrm{acc}}$	Classical ROF	Weight 1	Weight 2	Weight 3	Huber ROF
err $= 10^{-4}$	0.256	0.190	0.193	0.193	0.165
err $= 10^{-6}$	0.01	0.02	0.02	0.02	0.01
$\mathrm{d}_{L^2}^{\mathrm{acc}}$	Classical ROF	Weight 1	Weight 2	Weight 3	Huber ROF
err $= 10^{-4}$	0.005	0.007	0.008	0.008	0.003
err $= 10^{-6}$	0.01	0.01	0.01	0.01	0.00

Table 6: $f = \text{saw}$

$\mathrm{d}_{\mathrm{TV}}^{\mathrm{acc}}$	Classical ROF	Weight 1	Weight 2	Weight 3	Huber ROF
err $= 10^{-4}$	0.109	0.112	0.101	0.134	0.104
err $= 10^{-6}$	0.01	0.00	0.00	0.01	0.02
$\mathrm{d}_{L^2}^{\mathrm{acc}}$	Classical ROF	Weight 1	Weight 2	Weight 3	Huber ROF
err $= 10^{-4}$	0.005	0.011	0.017	0.021	0.004
err $= 10^{-6}$	0.00	0.00	0.00	0.01	0.00

As expected, the results of the accelerated and non-accelerated algorithm are very close in all measures, especially at the level of tolerance 10^{-6}. Therefore, from now on all the comparisons will be made on accelerated algorithms, as the results of the accelerated and non-accelerated algorithms are on average very similar and the number of iterations needed is significantly reduced.

We also notice the following phenomena, which will not be discussed later as they appear in a similar fashion for different choices of parameters. First, the parameter $\mathrm{d}_{L^2}^{\mathrm{noisy}}$ is measured in order to make a clearer comparison between the different methods—if the deviation from the noisy image were too large, maybe some parameters need to be offset to account for this. However, in all the experiments above they are of the same order of magnitude, so we will subsequently not present this variable. We also test for the parameter $d_{L^2}(\cdot, \mathrm{ROF})$, the normalized distance from the classical ROF minimizer; it is not presented in the above tables, as it is very small in all the experiments, which indicates that (as expected) the solution of the double-phase adaptive ROF problem is close in L^2 to the solution of the ROF problem. Moreover, the variable $\mathrm{d}_{L^2}^{\mathrm{image}}$ returns values of the same order of magnitude for all the algorithms—the parameter $\mathrm{d}_{\mathrm{TV}}^{\mathrm{image}}$ tracks the location and size of jumps much more accurately, so from now on we do not present the L^2 norm (we discuss this further in Section 8). Finally, we observe that for the tolerance level of 10^{-6} the accelerated and non-accelerated algorithms give very similar results, on the level 10^{-4} there are some differences—indeed, the accelerated algorithm at 10^{-4} is closer to the accelerated algorithm at 10^{-6} than to the standard algorithm at 10^{-4}.

3 Comparison for different values of the error parameter

We now report several comparisons between the results of the classical ROF minimization problem, the ROF problem with adaptive weights, and the Huber ROF problem, in relation to the choice of the parameter in the stop condition for the (accelerated) Chambolle–Pock algorithm. The corresponding file with the python code is named '1D-tolerance'. We present the results of the numerical experiment for the values of the error parameter in the set $\{10^{-3}, 10^{-4}, 10^{-5}, 10^{-6}, 10^{-7}\}$ for the same two test images as above and values of all other parameters, i.e., we choose the following parameters:

1. $\lambda = 0.24$;
2. $\sigma = 0.01$;
3. Radius of mollification: 2;
4. Weight 1: $a = 10^3$, $b = 10^4$;
5. Weight 2: $a = 10^3$, $b = 10^4$;
6. Weight 3: $h = 10^3$, $r = 0.03$.

We track the same measures as above, i.e., the (rescaled) gradient $\mathrm{d}_{\mathrm{TV}}^{\mathrm{image}}$ and the L^2 norm $\mathrm{d}_{\mathrm{L}^2}^{\mathrm{image}}$ of the difference of the result and the original image; the (rescaled) L^2 norm of the difference between the result and the noisy image $\mathrm{d}_{\mathrm{L}^2}^{\mathrm{noisy}}$; and the number of iterations. Below we present only the rescaled TV error $\mathrm{d}_{\mathrm{TV}}^{\mathrm{image}}$ and the number of iterations as the other metrics show very little variability.

Table 7: $f = \mathrm{jump}$

$\mathrm{d}_{\mathrm{TV}}^{\mathrm{image}}$	Classical ROF	Weight 1	Weight 2	Weight 3	Huber ROF
err $= 10^{-3}$	0.41	0.27	0.26	0.27	0.48
err $= 10^{-4}$	0.35	0.23	0.22	0.23	0.46
err $= 10^{-5}$	0.34	0.21	0.20	0.22	0.45
err $= 10^{-6}$	0.35	0.22	0.21	0.24	0.45
err $= 10^{-7}$	0.35	0.23	0.22	0.24	0.46
Iterations	Classical ROF	Weight 1	Weight 2	Weight 3	Huber ROF
err $= 10^{-3}$	204	210	225	225	126
err $= 10^{-4}$	719	523	549	549	242
err $= 10^{-5}$	2393	1205	1194	1200	662
err $= 10^{-6}$	7165	2880	2738	2737	1993
err $= 10^{-7}$	17372	7080	6903	6941	6112

Table 8: $f = \text{saw}$

d_{TV}^{image}	Classical ROF	Weight 1	Weight 2	Weight 3	Huber ROF
err $= 10^{-3}$	1.26	0.48	0.43	0.52	1.16
err $= 10^{-4}$	1.26	0.48	0.44	0.54	1.16
err $= 10^{-5}$	1.25	0.48	0.43	0.54	1.14
err $= 10^{-6}$	1.25	0.49	0.44	0.55	1.15
err $= 10^{-7}$	1.26	0.48	0.44	0.56	1.16
Iterations	Classical ROF	Weight 1	Weight 2	Weight 3	Huber ROF
err $= 10^{-3}$	188	178	207	212	126
err $= 10^{-4}$	612	424	524	539	260
err $= 10^{-5}$	1767	1213	1534	1560	776
err $= 10^{-6}$	4110	3638	4622	4637	2413
err $= 10^{-7}$	9108	11048	14074	13847	7694

In all the experiments, regardless of the type of the algorithm, in the presented range there is only a mild influence of the error parameter on the averaged measures. For the jump function, the distance to the original image is the largest for 10^{-3} (both in L^2 and its gradient), then it decreases and stabilizes in the range $10^{-4} - 10^{-7}$, while for the saw function the results are approximately the same for each level of tolerance. Similar results are replicated across different noise levels, values of λ, grid sizes, and parameters within weights (in the range discussed in a suitable section). Therefore, for further experiments we will run the algorithm at the tolerance level of 10^{-4} or 10^{-6}.

4 The mollification radius

We now study the performance of the algorithm with respect to the mollification radius during the computation of the weight. For this experiment, in all cases we again use the accelerated Chambolle–Pock algorithm. The corresponding file with the python code is named '1D-radii'. We present the results of the numerical experiment for the same two test images values of all other parameters as above, i.e., we choose the following parameters:

1. $\lambda = 0.24$;
2. $\sigma = 0.01$ and $\sigma = 0.001$;
3. Tolerance level: 10^{-4};
4. Weight 1: $a = 10^3$, $b = 10^4$;
5. Weight 2: $a = 10^3$, $b = 10^4$;
6. Weight 3: $h = 10^3$, $r = 0.03$.

We present the results in Figures 1-2; we plot only d_{TV}^{image}, i.e., the gradient of the difference of the result and the original image (the available code produces all the tested measures).

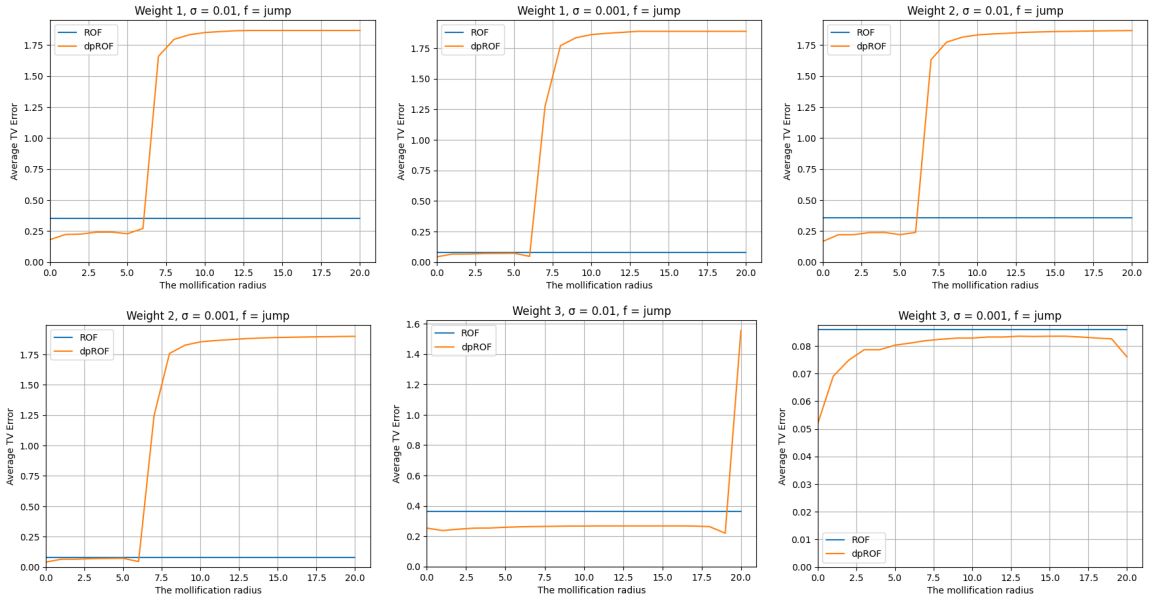

Figure 1: Comparison of radii: jump function

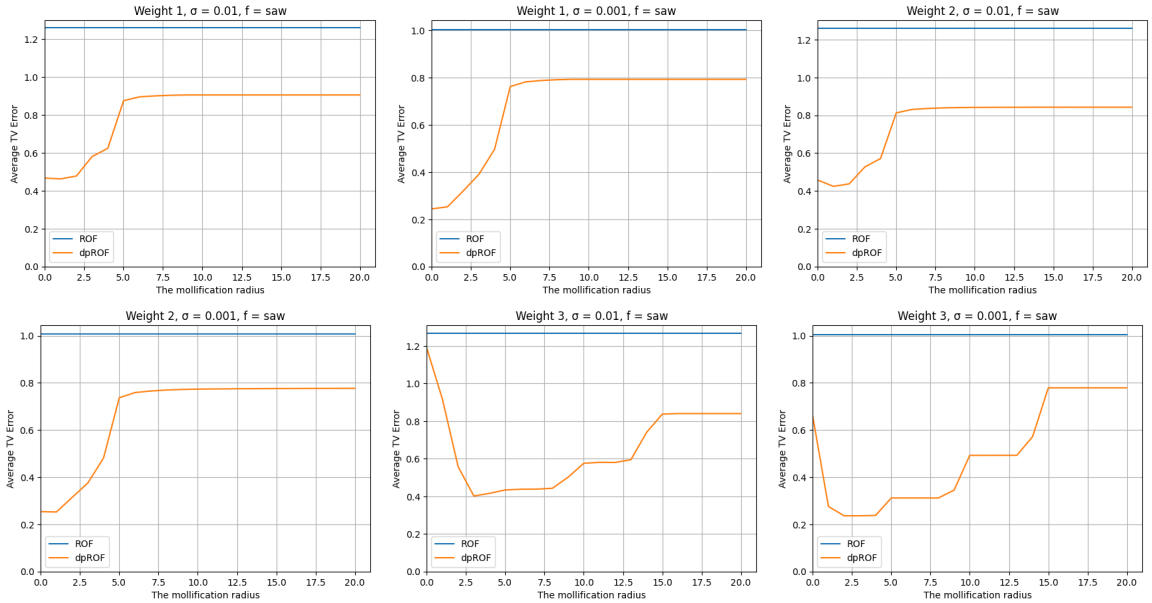

Figure 2: Comparison of radii: saw function

We observe the following effects. First, for most of the tested images and weights, there is some benefit in smoothing the initial data (this is most visible for Weight 3). We also note that across different choices of the radius, the results for Weight 1 and Weight 2 for the same parameters are nearly identical. Moreover, the effect of mollification is different for different images: for the function "jump", which has only jump-type derivative, the measure $\mathrm{d}_{\mathrm{TV}}^{\mathrm{image}}$ blows up as the radius increases; and for the function "saw", which includes both continuous parts and jumps of different sizes, the result gets worse as the radius increases, but it is still closer to the original image than the result of the classical ROF. These observations are replicated over different noise levels. Therefore, it seems it is best to use the algorithm with radii between two and five; in the subsequent experiments, we will use radius two or three.

5 Comparison of different λ

We now study the performance of the algorithm with respect to the mollification radius during the computation of the weight. For this experiment, in all cases we again use the accelerated Chambolle–Pock algorithm. The corresponding file with the python code is named '1D-lambdas'. We present the results of the numerical experiment for the same two test images values of all other parameters as above, i.e., we choose the following parameters:

1. $\sigma = 0.01$ and $\sigma = 0.001$;
2. Tolerance level: 10^{-4};
3. Mollification radius: 2;
4. Weight 1: $a = 10^3$, $b = 10^4$;
5. Weight 2: $a = 10^3$, $b = 10^4$;
6. Weight 3: $h = 10^3$, $r = 0.03$.

We present the results in Figures 3-4; we plot only $\mathrm{d}_{\mathrm{TV}}^{\mathrm{image}}$, i.e., the normalized gradient of the difference of the result and the original image (the available code produces all the tested measures).

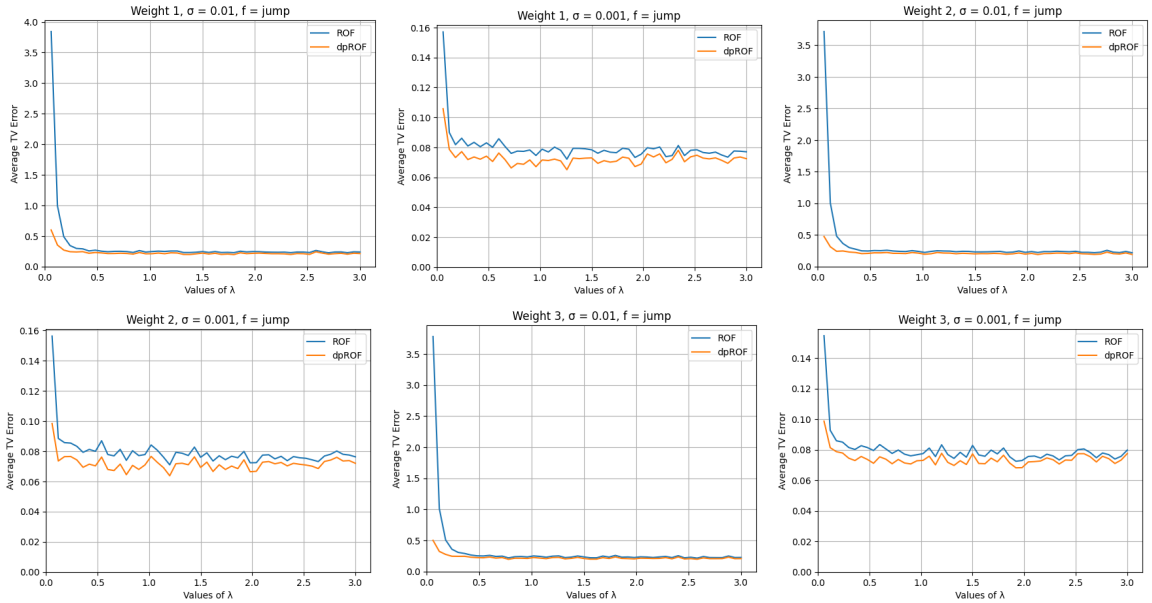

Figure 3: Comparison of λ's: jump function

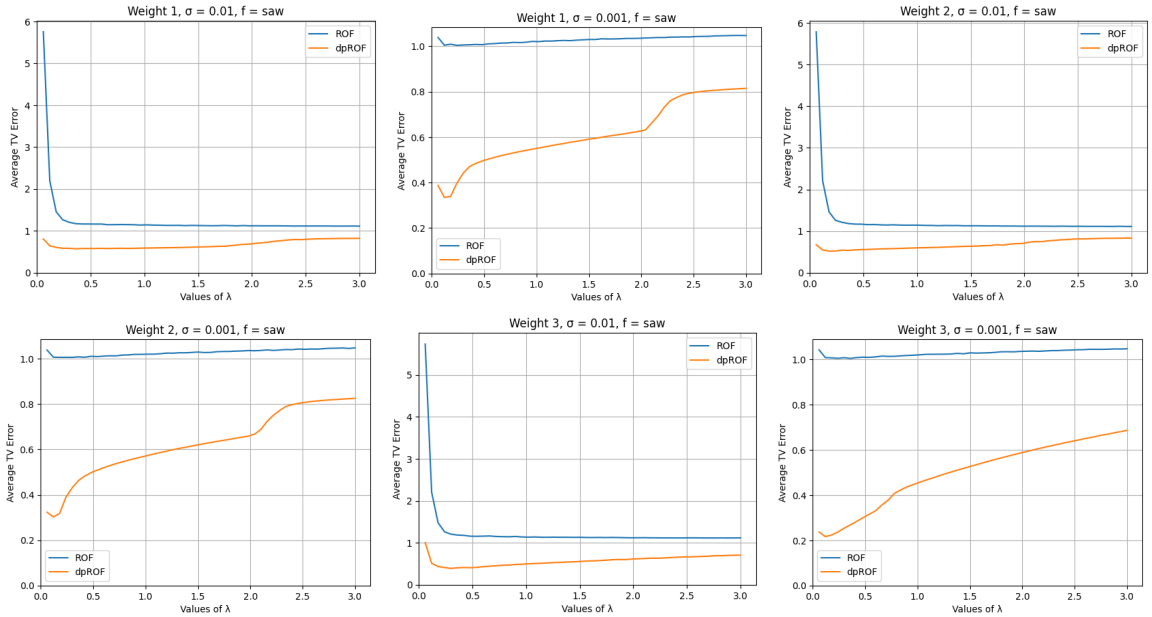

Figure 4: Comparison of λ's: saw function

6 Comparison of different weights

We now study the performance of the algorithm with respect to the different parameters within each studied type of the weight. For this experiment, in all cases we use the accelerated Chambolle–Pock algorithm. The corresponding files with the python code are named '1D-weights12' (for Weights 1 and 2) and '1D-weights3' (for Weight 3).

We split this presentation into two parts: for "synthetic" images with clear-cut features and for "natural" images, i.e., one-pixel stripes of real images, with distinctive features across different scales. It turns out that finding optimal values of the weight for practical use varies between these types of images. In line with the previous observation that using Weight 1 and Weight 2 leads to nearly identical results, we present the results only for Weight 1 and Weight 3. For all the images, we choose the following parameters:

1. $\lambda = 0.24$;
2. $\sigma = 0.01$;
3. Tolerance level: 10^{-4};
4. Grid size: 1000, 343, 1024 or 4032.

We again present only the rescaled gradient of the difference of the result and the original image. In Figures 5–14 we show the results for the two usual images 'jump' and 'saw', as well as two one-dimensional stripes of a real image 'cut1', 'cut2' and 'cut3' (included in the files bundle).

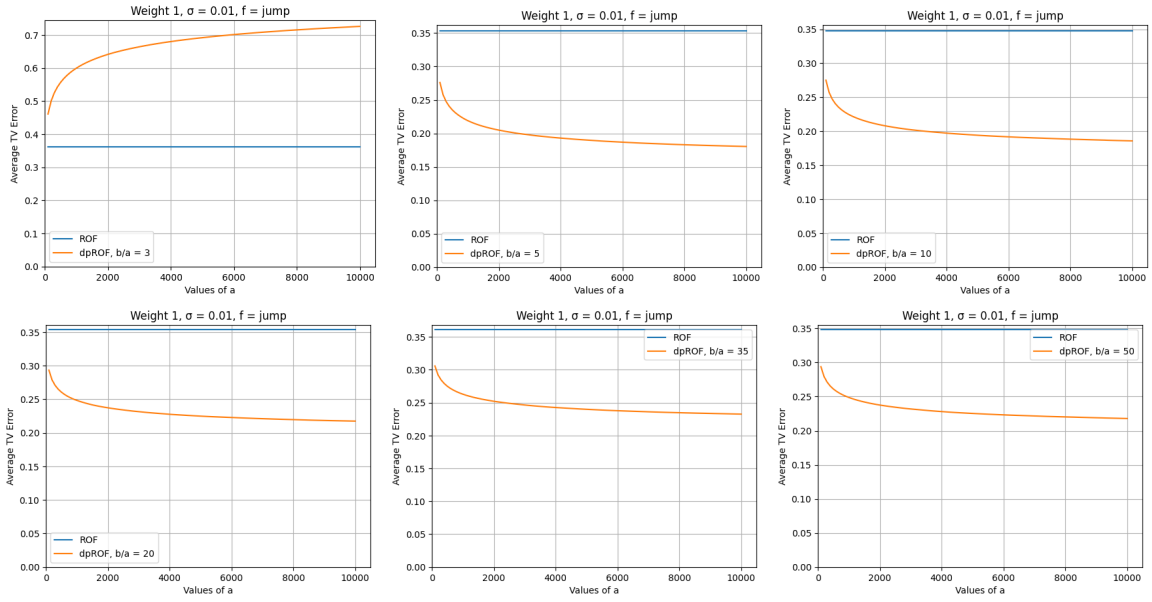

Figure 5: Comparison of parameters: Weight 1, jump function

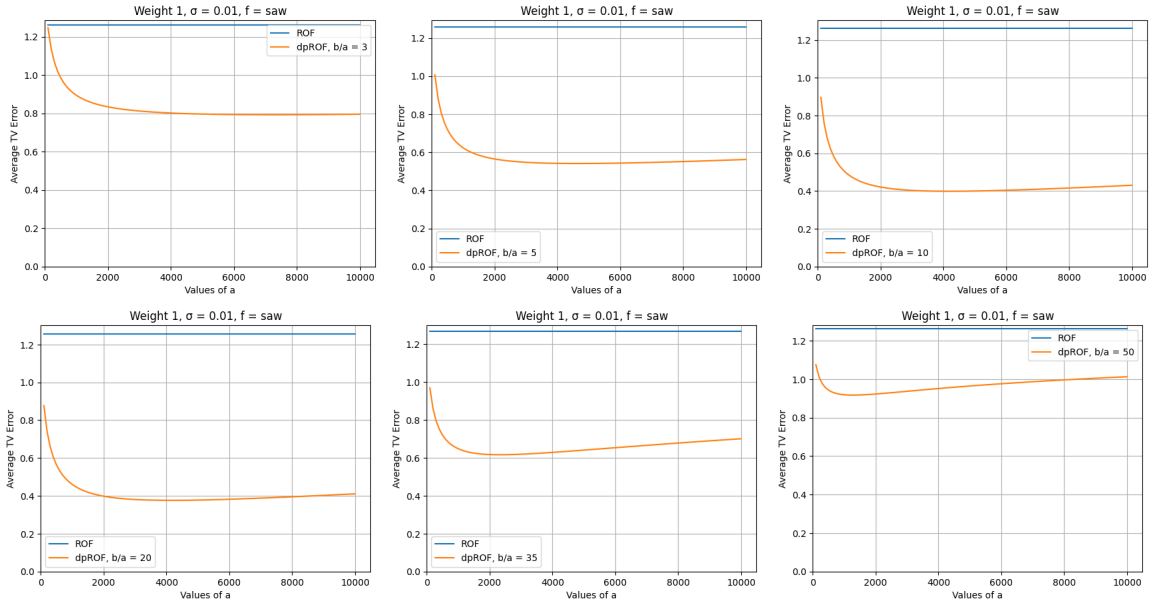

Figure 6: Comparison of parameters: Weight 1, saw function

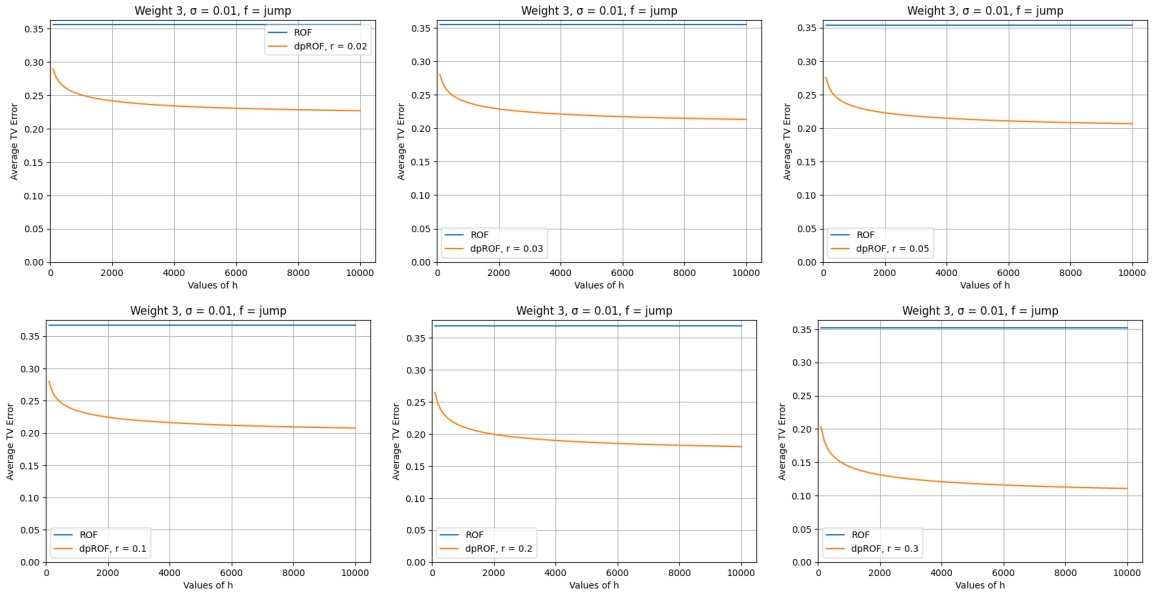

Figure 7: Comparison of parameters: Weight 3, jump function

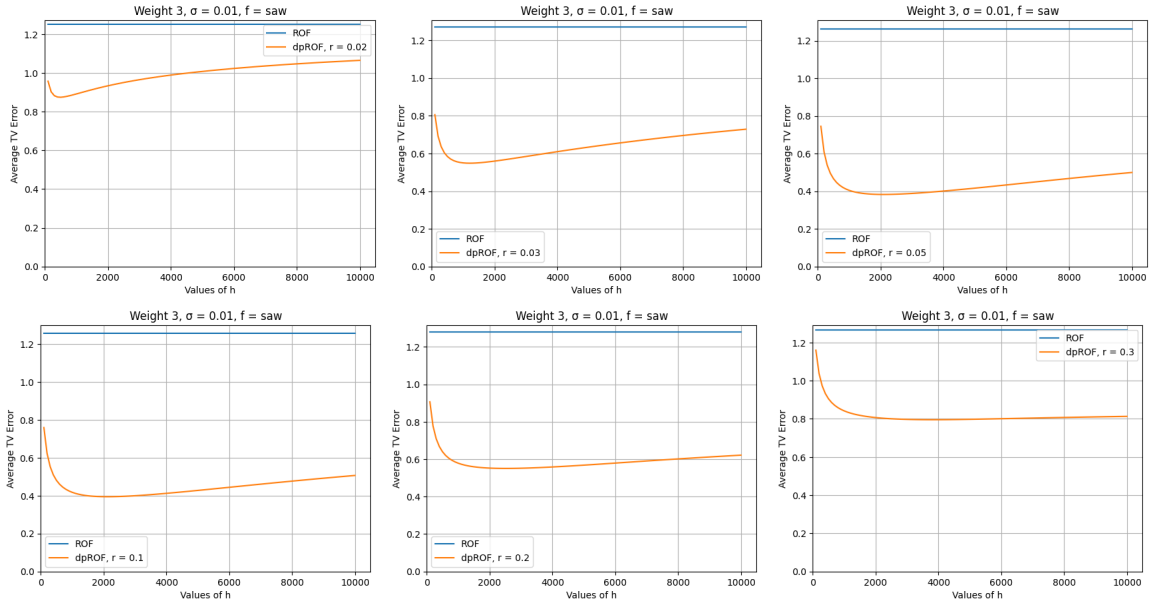

Figure 8: Comparison of parameters: Weight 3, saw function

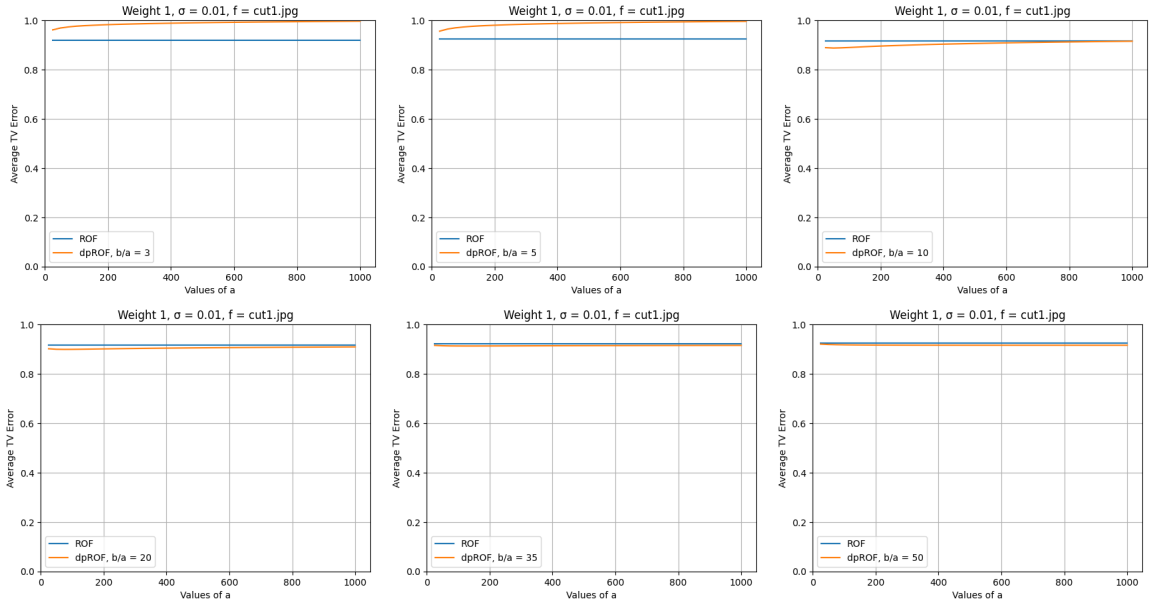

Figure 9: Comparison of parameters: Weight 1, cut1 function

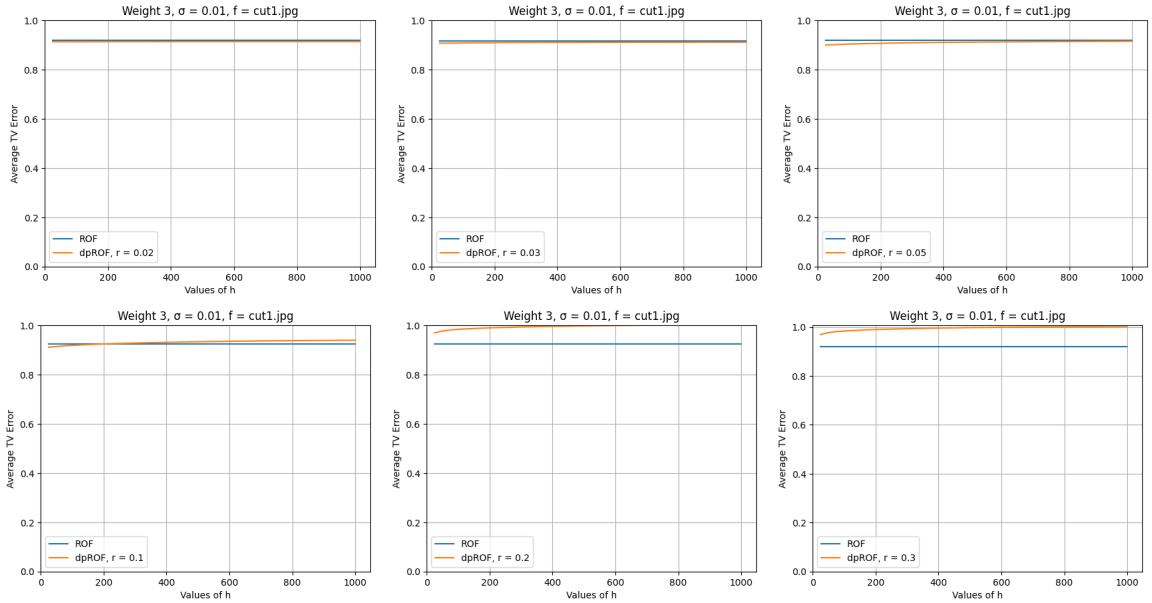

Figure 10: Comparison of parameters: Weight 3, cut1 function

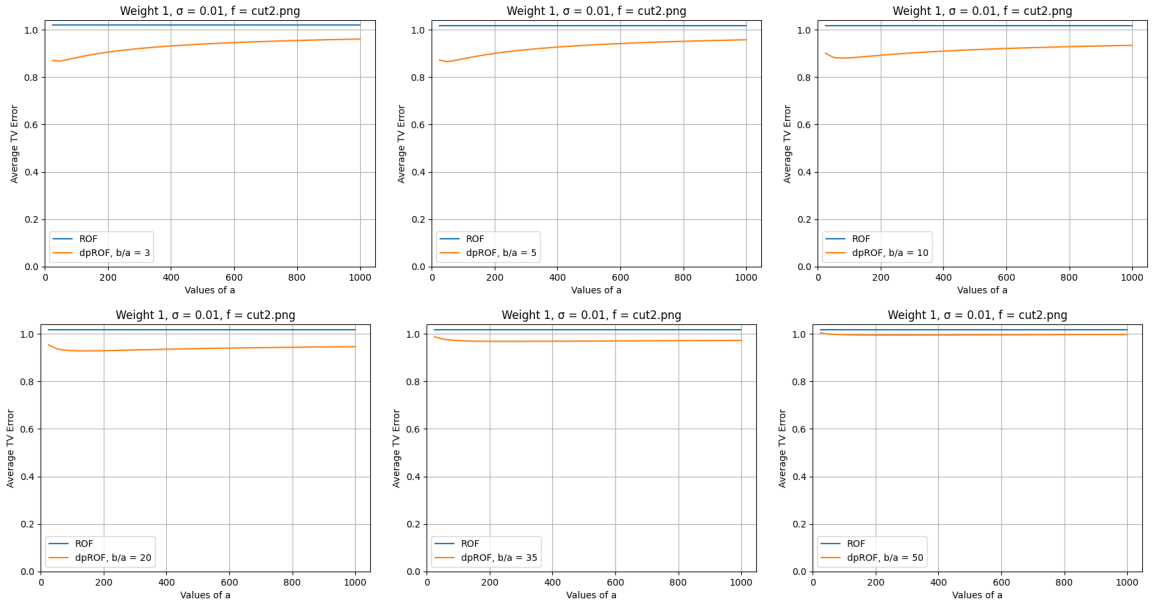

Figure 11: Comparison of parameters: Weight 1, cut2 function

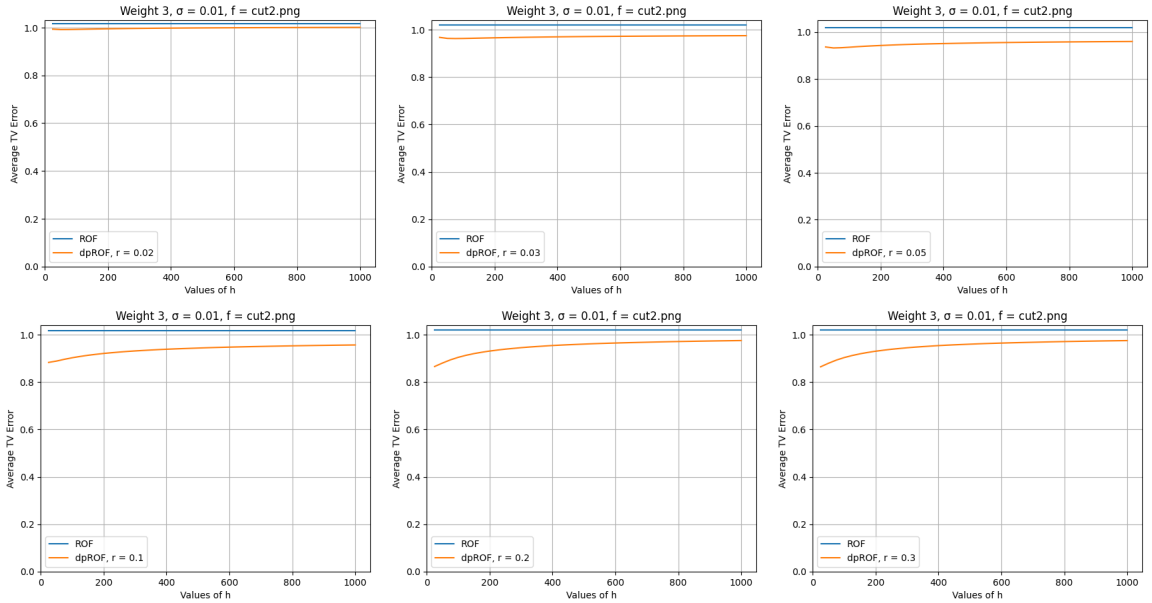

Figure 12: Comparison of parameters: Weight 3, cut2 function

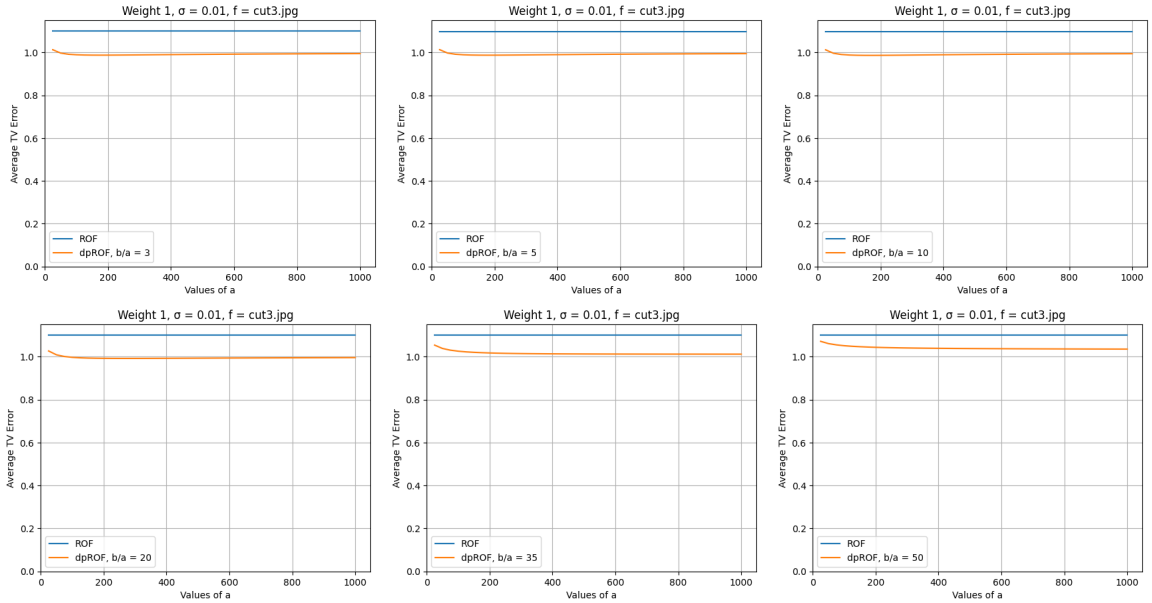

Figure 13: Comparison of parameters: Weight 1, cut3 function

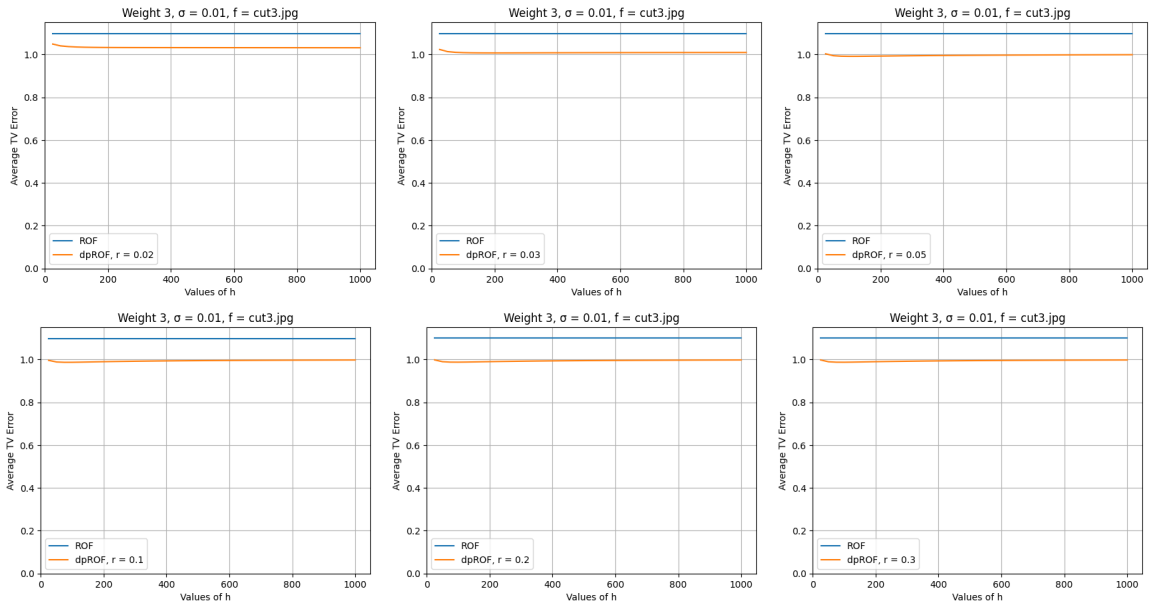

Figure 14: Comparison of parameters: Weight 3, cut3 function

7 Different noise levels

We now take the optimized weights from the previous section and study the performance of the algorithm with respect to the different noise levels. For this experiment, in all cases we use the accelerated Chambolle–Pock algorithm. The corresponding files with the python code are named '1D-noise'. For all the images, we choose the following parameters:

1. $\lambda = 0.24$;
2. $\sigma = 0.01$;
3. Tolerance level: 10^{-4};
4. Grid size: 1000, 343, 1024 or 4032.

We choose different coefficients in weights for "synthetic" and "natural" images. We again present only the gradient of the difference of the result and the original image; this is done in Figures 15–18. For the 'jump' and 'saw' image, we use the parameters $a = 10^3$, $b = 10^4$ (Weight 1) or $h = 10^3$, $r = 0.05$ (Weight 3), while for the 'cut1', 'cut2' and 'cut3' images we use the parameters $a = 20$, $b = 400$ (Weight 1) or $h = 400$, $r = 0.05$ (Weight 3).

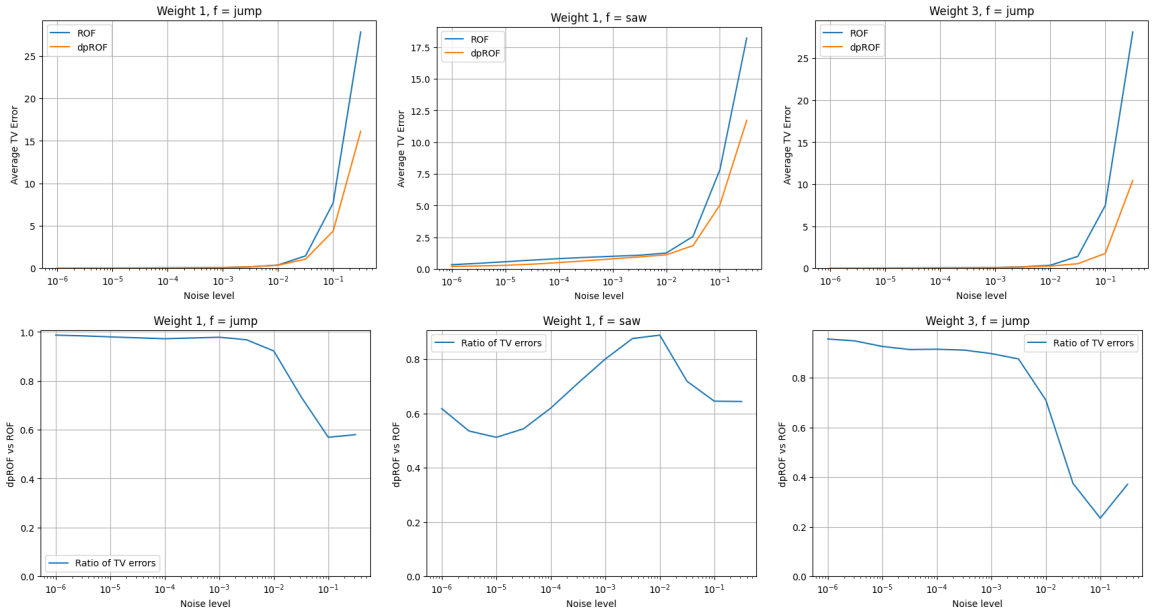

Figure 15: Different noise levels, part I

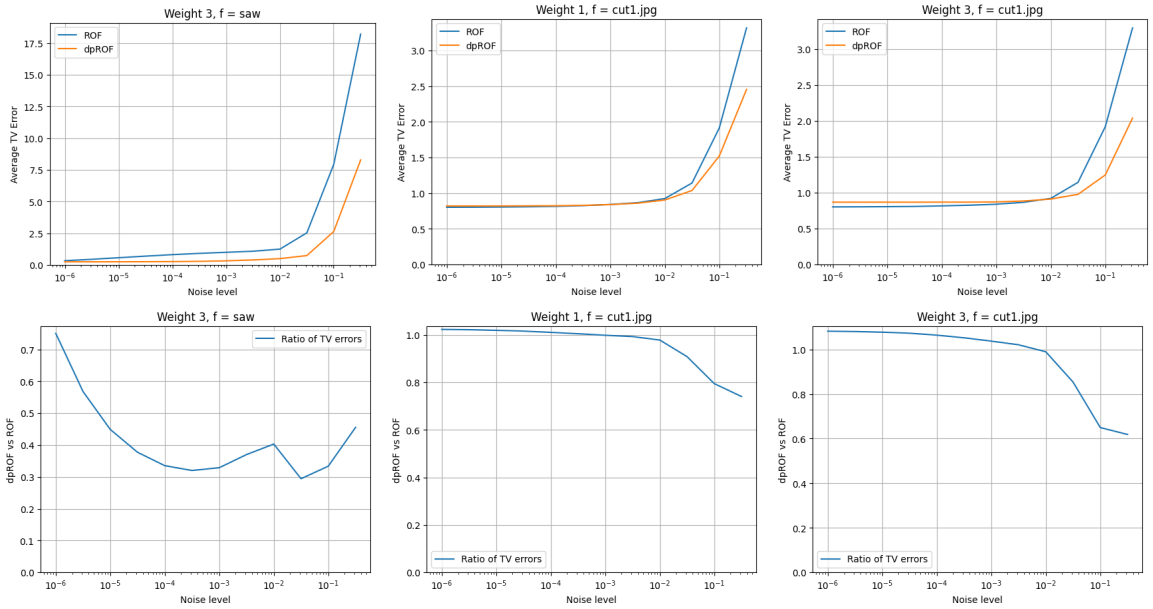

Figure 16: Different noise levels, part II

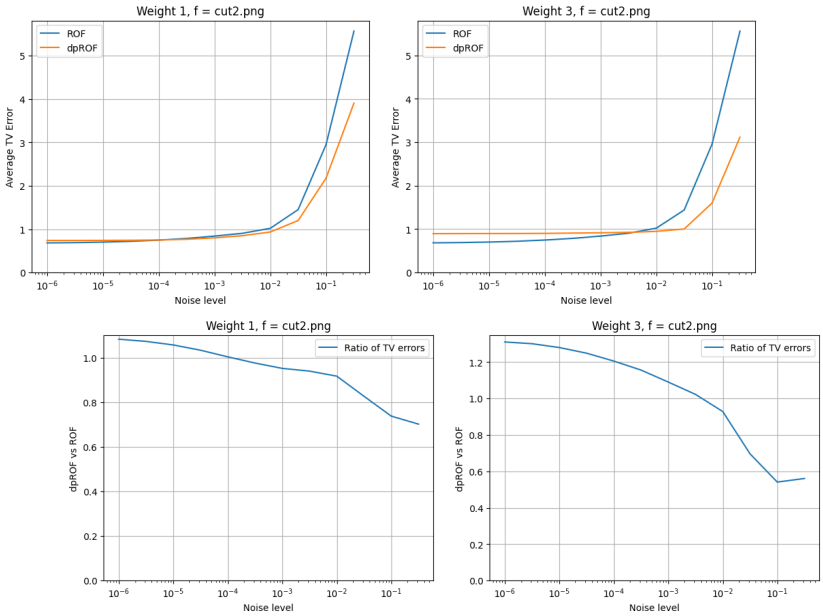

Figure 17: Different noise levels, part III

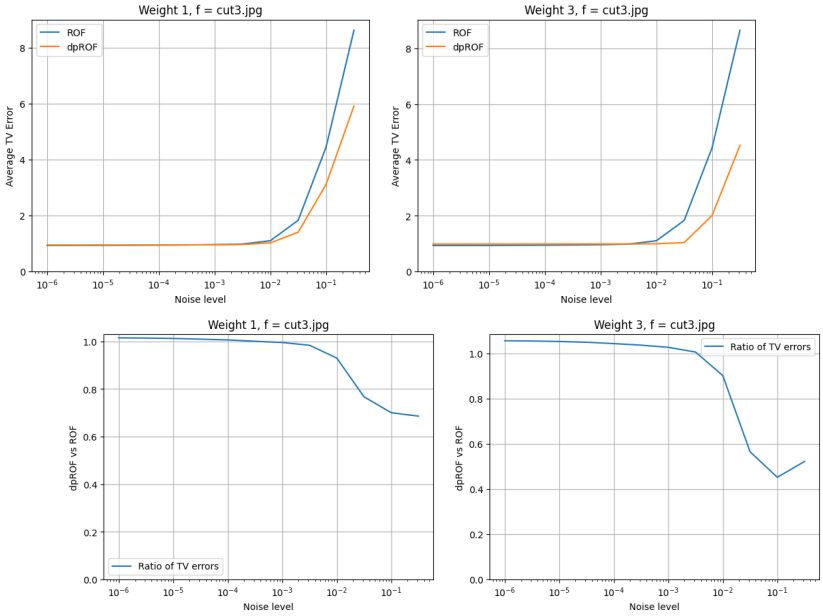

Figure 18: Different noise levels, part IV

8 The normalized L^2 distance

Let us briefly comment on the second tested measure, the normalized L^2 distance to the original image, i.e.,

$$\mathrm{d}_{L^2}^{\mathrm{image}}(\mathrm{result}) = \frac{\|\mathrm{result} - \mathrm{image}\|_2}{\|\mathrm{image}\|_2}.$$

This measure behaves similarly to the normalized TV distance $\mathrm{d}_{TV}^{\mathrm{image}}$, but shows less variability. This is indicative of the fact that the measure $\mathrm{d}_{TV}^{\mathrm{image}}$ is better suited to capture staircasing phenomena for 1D images—formation of sharp staircases, which moves the location of the jumps by several pixels, has little effect on the L^2 norm but significantly changes the TV error. We present for comparison a selection of plots corresponding to ones from the previous two Sections.

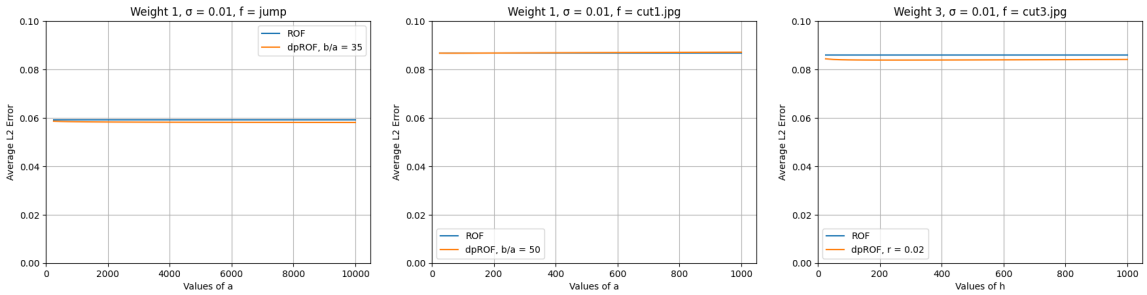

Figure 19: Different weights - L^2 distance

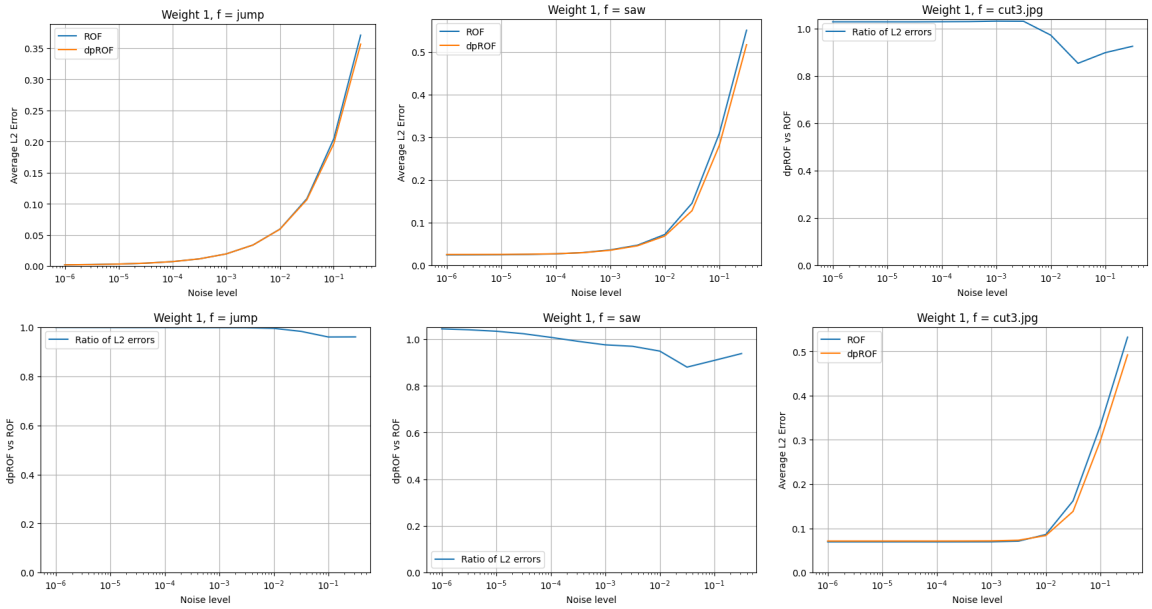

Figure 20: Different noise levels: L^2 distance

9 Comparison with the Huber ROF model

Choosing different values of the parameter λ in the classical ROF model corresponds to choosing different values of the L^2-distance to the noisy image in the original formulation. To compare the performance of the classical ROF model, the adaptive double-phase ROF model and the Huber ROF model, we present the results for the normalized TV error $d_{\mathrm{L}^2}^{\mathrm{image}}$ and the PSNR metric in terms of the (normalized) L^2-distance to the noisy image.

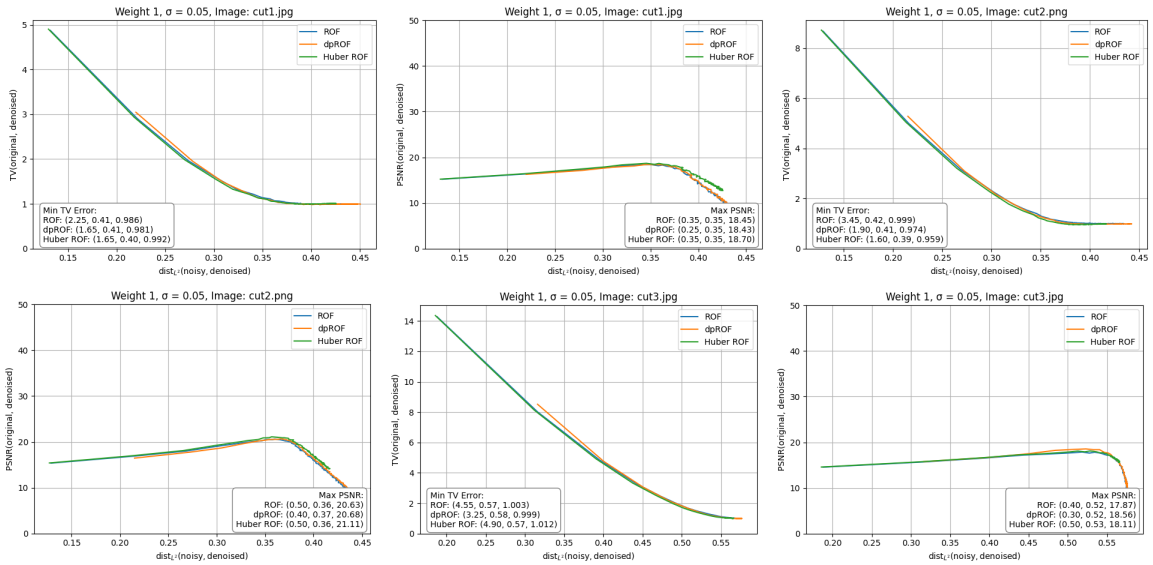

Figure 21: Comparison with Huber ROF: part one

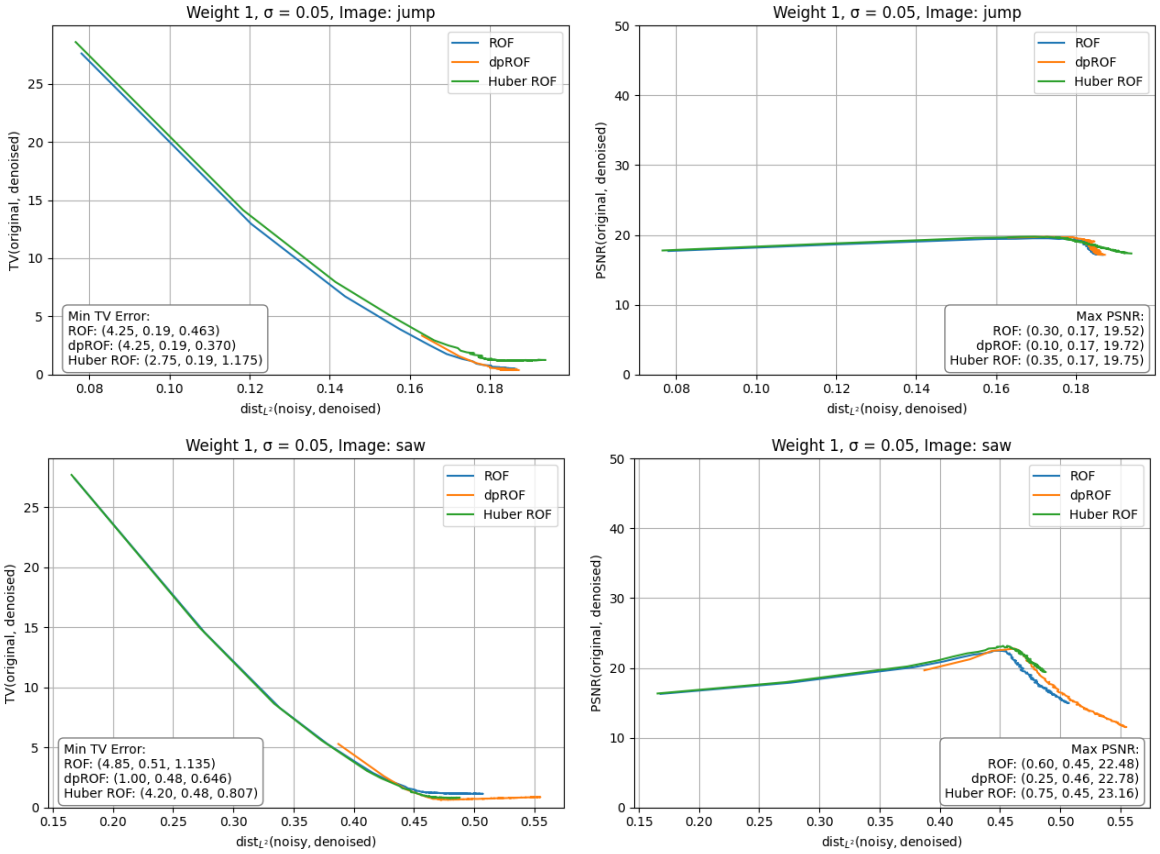

Figure 22: Comparison with Huber ROF: part two

10 Plots

We now present visually the effect of our algorithm on several test functions, both "synthetic" and "natural". In each of the presented pictures, we first show the original image; the noisy image; the effect of the classical ROF model; the effect of the adaptive double-phase ROF model with four choices of parameters within Weight 1; and the result of the Huber ROF model (with parameter 0.001). In each case, we use the following parameters:

1. $\lambda = 0.24$;
2. $\sigma = 0.01$;
3. Tolerance level: 10^{-6};
4. Grid size: 1000, 343, 1024 or 4032.

In Figure 23, we present the results for the 'saw' function. We notice a strong reduction of staircasing with respect to the classical ROF model, and (up to rescaling the image) for large values of the weights we have a nearly perfect reconstruction of the original image.

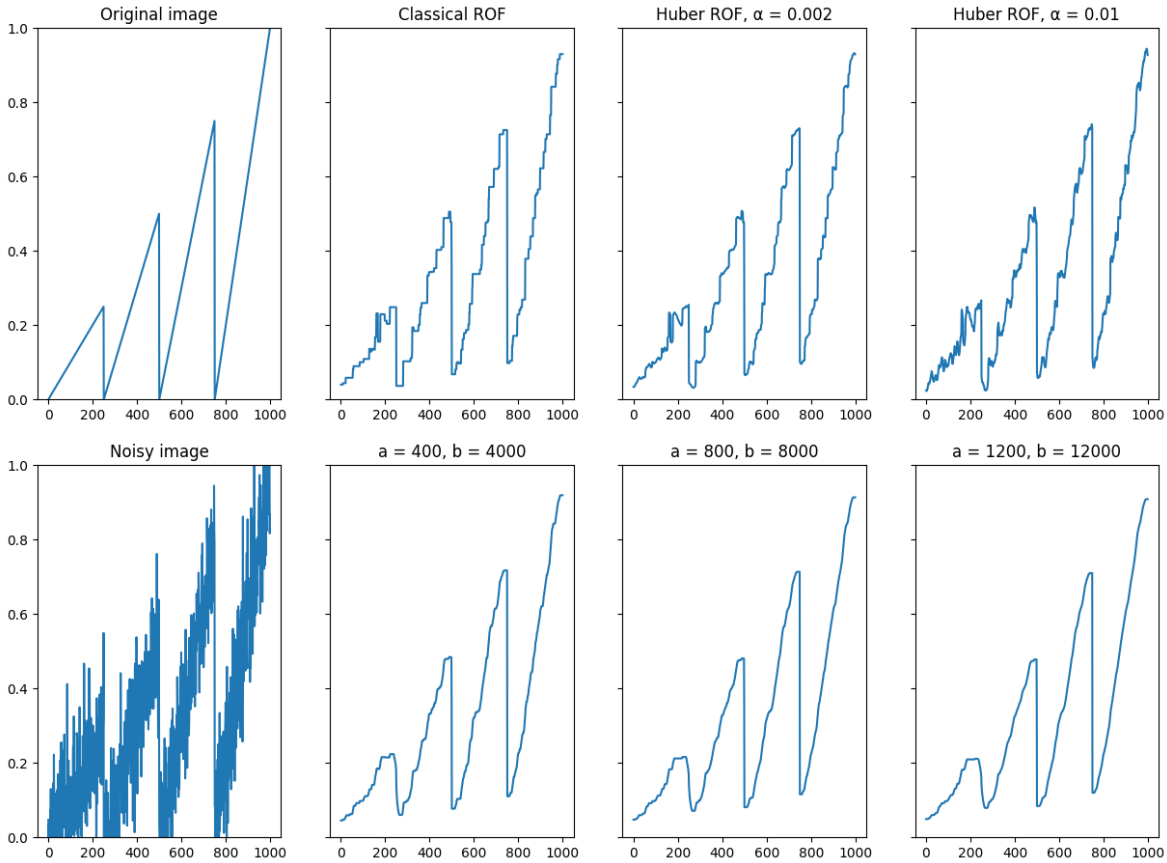

Figure 23: The saw function

Note that, due to the nature of the model, large distortions of the original image present in the result of the classical ROF model are also present here as the weight is computed from the solution of the standard ROF minimization; this is most visible in the left-hand side of the graphs. The result of the Huber ROF minimization also shows a reduction of staircasing, but the reconstructed image is less similar to the original image with respect to the adaptive double-phase ROF model. Similar behavior is shared by other synthetic images, such as 'barcode' (presented in the top left corner of Figure 24). In Figure 24, we also observe a nearly perfect reconstruction with a large reduction of staircasing with respect to the classical ROF model.

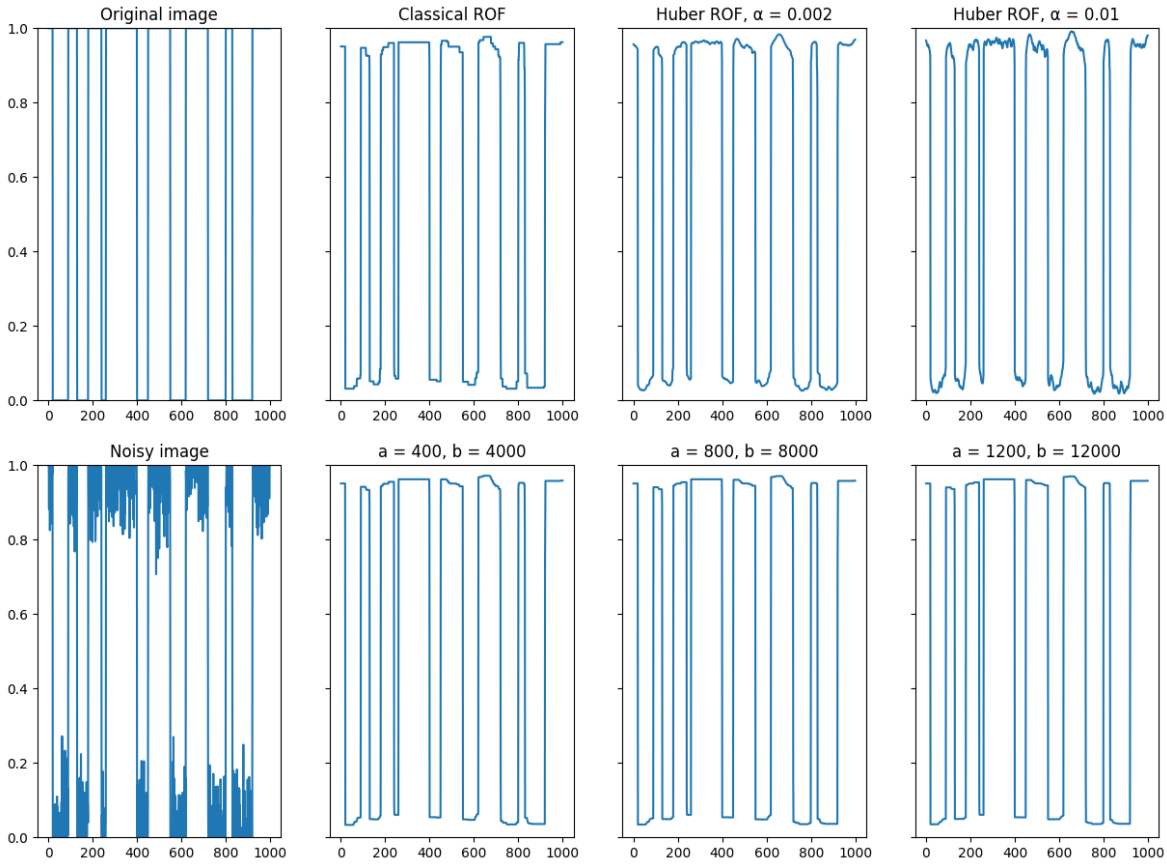

Figure 24: The barcode function

We observe also the above behavior for natural images, such as 'cut1', 'cut2' and 'cut3' presented below, with the following caveat. For images which show a large variability on small scales, the original ROF model already destroys many features of the original image; since we rely on the result of the classical ROF minimization for the computation of the weight, this naturally extends to the adaptive double-phase ROF model. Thus, in Figures 25-27 which present the 'cut1' (rescaled by a factor of 2), 'cut2' and 'cut3' functions respectively, we observe a significant reduction in staircasing with respect to the classical ROF model, while maintaining a similar level of accuracy in the reconstruction of the original image. The level of accuracy increases with the size of the grid, while the reduction of staircasing is readily seen at all scales. The results of the Huber ROF minimization are now quite close to the results of the adaptive double-phase ROF model, with the latter showing a slightly bigger reduction of staircasing. The results exhibit similar behavior for different choices of the noise level and (within reasonable range for classical ROF) of λ.

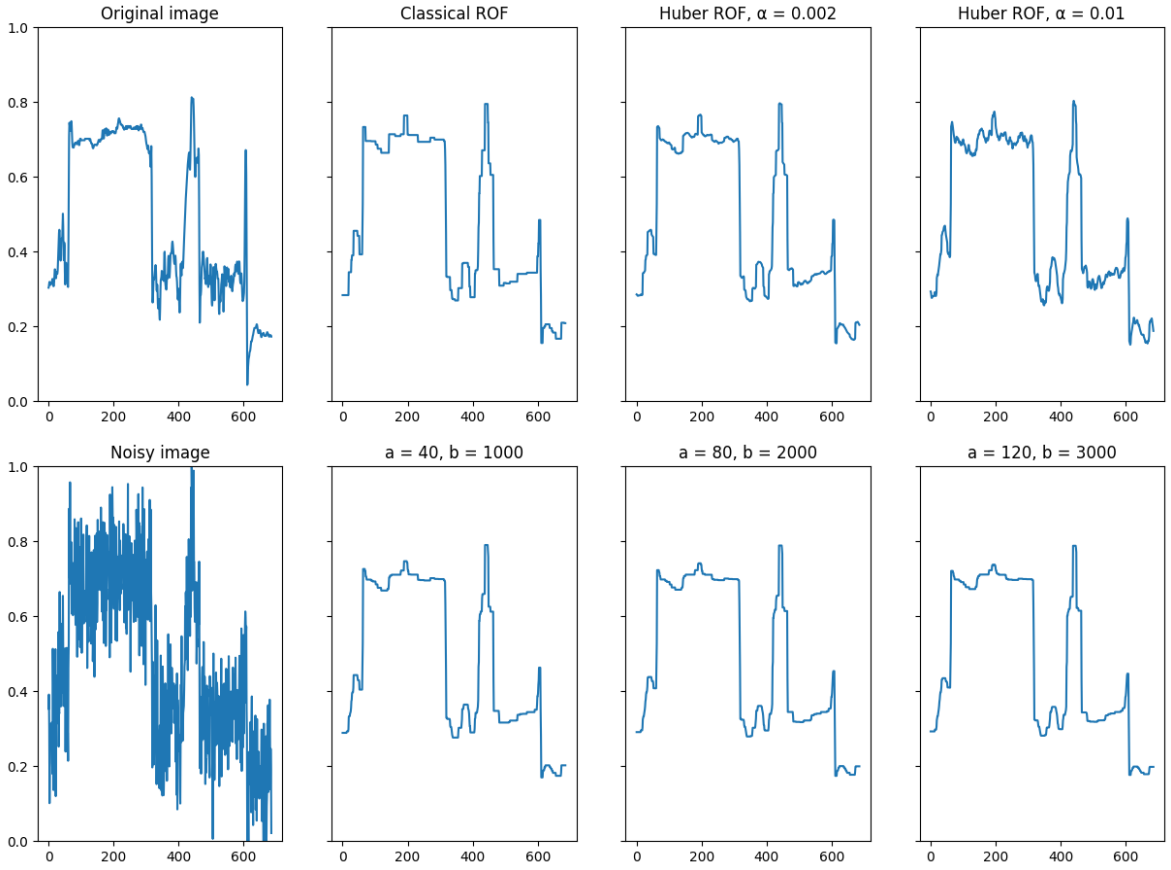

Figure 25: The cut1 function

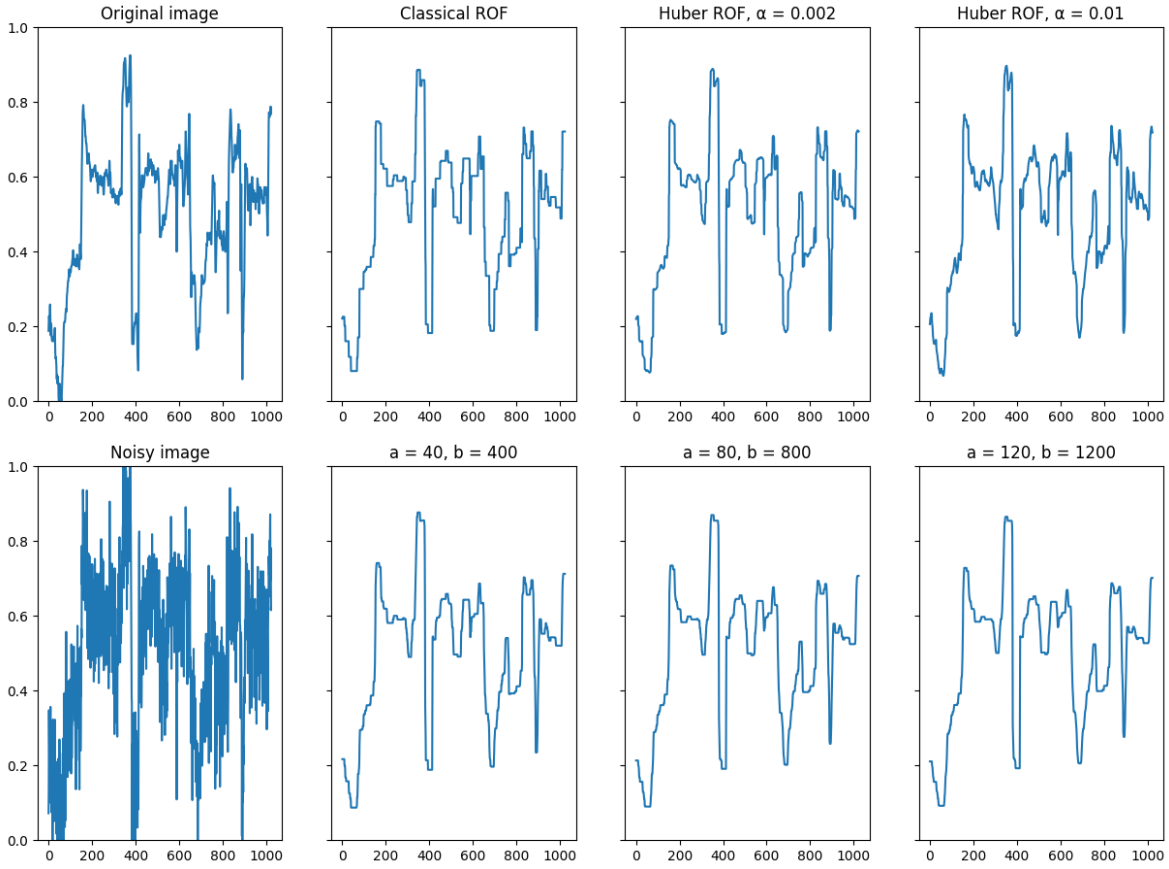

Figure 26: The cut2 function

Funding. The work of the first author has been supported by the Austrian Science Fund (FWF), grants 10.55776/ESP88 and 10.55776/I5149. The second author has been partially supported by the grant 2024/55/D/ST1/03055 of the National Science Centre (NCN), Poland. The third author has been partially supported by the Special Account for Research Funding of the National Technical University of Athens.

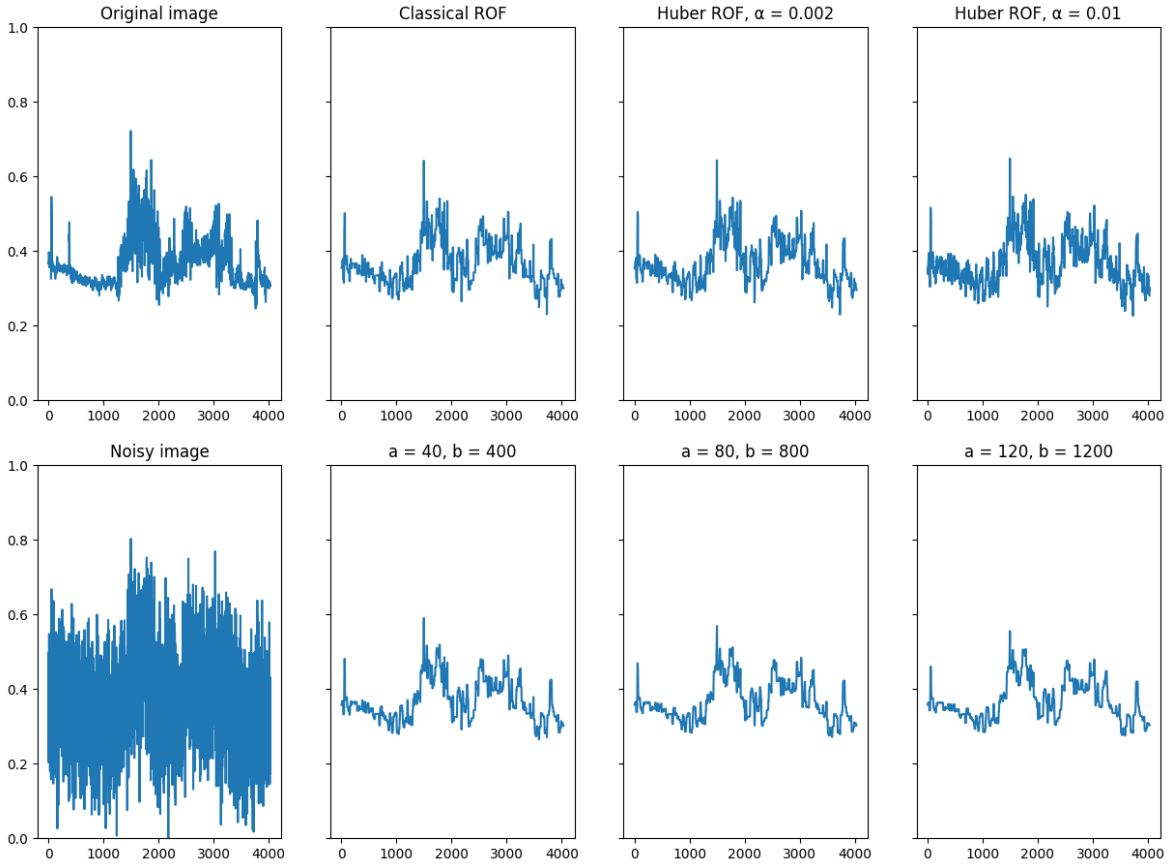

Figure 27: The cut3 function

References

[1] Antonin Chambolle, Thomas Pock. A first-order primal-dual algorithm for convex problems with applications to imaging. *J. Math. Imaging Vis.* 40 (2011) pp. 120–145. doi: 10.1007/s10851-010-0251-1.

[2] Wojciech Górny, Michał Łasica, Alexandros Matsoukas. Euler–Lagrange equations for variable-growth total variation (preprint 2025). `arXiv:2504.13559`.

[3] Petteri Harjulehto, Peter Hästö. Double phase image restoration. *J. Math. Anal. Appl.* 501 (2021). doi:10.1016/j.jmaa.2019.123832.

[4] Leonid I. Rudin, Stanley Osher, Emad Fatemi. Nonlinear total variation based noise removal algorithms. *Phys. D* 60 (1992) pp. 259–268. doi:10.1016/0167-2789(92)90242-F. Experimental mathematics: computational issues in nonlinear science (Los Alamos, NM, 1991).