L1-Norm Redundant Delaunay Phase Unwrapping and Gradient Correction

Alexandre Achard-de Lustrac, Roland Akiki, Axel Davy, Jean-Michel Morel

Published in Image Processing On Line on YYYY–MM–DD.
Submitted on YYYY–MM–DD, accepted on YYYY–MM–DD.
ISSN 2105–1232 © YYYY IPOL & the authors CC–BY–NC–SA
This article is available online with supplementary materials,
software, datasets and online demo at
https://ipolcore.ipol.im/demo/clientApp/demo.html?id=77777000528

L_1-Norm Redundant Delaunay Phase Unwrapping and Gradient Correction

Alexandre Achard-de Lustrac[1], Roland Akiki[2], Axel Davy[1], Jean-Michel Morel[3]

[1] Université Paris-Saclay, ENS Paris-Saclay, Centre Borelli, F-91190 Gif-sur-Yvette, France
alexandre.achard-de-lustrac@universite-paris-saclay.fr
axel.davy@ens-paris-saclay.fr
[2] Kayrros SAS
roland.akiki@ens-paris-saclay.fr
[3] City University of Hong Kong
jeamorel@cityu.edu.hk

PREPRINT February 7, 2025

Abstract

This article deals with arrays of real numbers which have been reduced modulo $2h$ into the interval $[-h, h]$ where $h > 0$ is a positive real number. Such an array is said to be wrapped modulo $2h$. Often, the elements of these arrays correspond to values observed at points in an image-like 2D space which are connected by a graph structure. The process of retrieving the original array from which the wrapped image originates is called unwrapping. Of course, the wrapping process is not one-to-one, and the quality of the recovered unwrapped version depends on the smoothness of the original array. The goal of unwrapping is to define a most plausible left inverse (as will be defined in a precise way) to the non-injective modulation operator mod $2h$ using heuristic arguments and regularity assumptions on the original signal. Following the guidelines described in [3] and [5], this is made possible by correcting an approximate gradient into a global gradient using either linear programming or, in some cases, minimum-cost flow techniques to solve an L_1-norm optimization problem. Such a gradient-correcting technique can also be used in general for finding a most plausible gradient and reconstructing a signal. The online demo associated with this paper implements the aforementioned methods.

Source Code

The reviewed source code and documentation for this algorithm are available from the web page of this article[1]. Usage instructions are included in the `README.txt` file of the archive.

Keywords: InSAR; Persistent Scatterers; Phase unwrapping; Gradient correction; Integration.

[1] https://ipolcore.ipol.im/demo/clientApp/demo.html?id=77777000528

1 Introduction

1.1 General framework

Phase unwrapping occurs in specialized image recovery fields such as InSAR (satellite imaging [3]) or MRI (medical imaging [16]). The unwrapped numbers can generally be interpreted as phases accumulated in electromagnetic waves during their travel. Phases being real numbers reduced modulo 2π, $h \coloneqq \pi$ denotes the half-modulus, hence the name *phase unwrapping*. Since phase and wavelength are related in electromagnetic waves, h can nevertheless be different from π and construed as a characteristic length. Interferometric Synthetic Aperture Radar or InSAR is a technique used in the reconstruction of elevation profiles or to monitor terrain deformation over time. It consists in sending radiation in the microwave domain of the electromagnetic spectrum from a satellite onto the Earth, and recovering the phase of the reflected light. The choice of these wavelengths prevents atmospheric absorption, so that the method remains functional even in cloudy areas. The recovered phase is a real number lying inside the normalized interval $[-\pi,\pi)$. However, during its travel, the phase of the recovered electromagnetic wave has completed several cycles of length 2π which have been cast out from the observable phase by a modulation mod 2π. Since the real phase before the wrapping process mod 2π is directly related to the distance (a full cycle representing the span of a full wavelength), undoing this process (i.e. phase unwrapping) would allow one to recover the elevation profile under scrutiny. In practice, differences of wrapped images, or interferograms, are considered to recover terrain deformation. To limit the effects of observation noise, several advanced differential interferometry techniques restrict the phase observations to an irregularly sampled planar set of points called "persistent scatterers" or PSs. Each PS is associated a real number in $[-\pi,\pi)$ called the wrapped phase ([8],[9]). PSs are usually linked to their spatial neighborhoods by edges in a graph. Such a graph can be defined by a Delaunay triangulation, possibly with some additional redundancy. By convention, the redundancy r is a nonnegative integer, where $r = 0$ corresponds to the default Delaunay triangulation and $r > 0$ corresponds to the default Delaunay triangulation plus the additional edges directly linking the k-th nearest neighbors of every vertex in the Delaunay triangulation for $1 \leq k \leq r + 1$ (redundant edges between two nodes are counted only once). As will be discussed later, the Delaunay triangulation can be degenerate (i.e. having triangles with zero surface area) without causing any problems. Empirically, this often happens when large numbers of points are located within a small compact area of the plane with some degree of regularity, but this is no cause of concern, as long as one uses joggled input (the input points are slightly joggled before triangulating by displacing them randomly within a small neighboring area; this ensures that the produced default Delaunay graph is planar). In Scipy's Delaunay implementation, this corresponds to option QJ which is passed to library Qhull.

Among the graphs under consideration, two categories can be delineated. The first category comprises planar graphs, i.e. graphs which can be drawn in the 2D Euclidean plane with edges not crossing each other except, perhaps, at their extremities (default Delaunay graphs with $r = 0$ and option QJ, possibly degenerate) while the second category comprises essentially non-planar graphs (Delaunay triangulations with redundancy $r > 0$). This distinction is important, because two different methods will be introduced, a minimum-cost flow method (MCF) which is more efficient but requires graphs in the first category, and a more general method using linear programming (LP) which applies to all graphs but is less efficient. All the graphs under consideration will be assumed to be connected.

To remain consistent with the concept of *phase* unwrapping, the convention $h \coloneqq \pi$ shall henceforth be adopted, but all the reasoning remains correct if one replaces π with any real $h > 0$.

1.2 The essential idea

Let us now suppose we have a set of PSs in a 2D plane with a graph. The main idea underpinning phase unwrapping is that phase differences between two vertices linked by an edge are relatively small (strictly smaller than π in absolute value). This is why only close points are being linked, since it is assumed that the underlying unwrapped signal is relatively smooth. Suppose $\overline{\varphi_x}$ and $\overline{\varphi_y}$ are the wrapped phases of two linked points x, y. Under the previously mentioned smoothness hypothesis, a good and mostly plausible approximation of the unwrapped phase difference can be given as $\mathcal{W}(\overline{\varphi_y} - \overline{\varphi_x})$ where $\mathcal{W}$ denotes the wrapping operator mod 2π in $[-\pi, \pi]$. More precisely, since by definition $\overline{\varphi_y} - \overline{\varphi_x} \in (-2\pi, 2\pi)$, we have

$$\mathcal{W}(\overline{\varphi_y} - \overline{\varphi_x}) = \begin{cases} \overline{\varphi_y} - \overline{\varphi_x} - 2\pi & \overline{\varphi_y} - \overline{\varphi_x} > \pi \\ \overline{\varphi_y} - \overline{\varphi_x} & \overline{\varphi_y} - \overline{\varphi_x} \in [-\pi, \pi] \\ \overline{\varphi_y} - \overline{\varphi_x} + 2\pi & \overline{\varphi_y} - \overline{\varphi_x} < -\pi. \end{cases} \tag{1}$$

The reason we wrap in $[-\pi, \pi]$ instead of $[-\pi, \pi)$ is to ensure the antisymmetric property

$$\mathcal{W}(\overline{\varphi_y} - \overline{\varphi_x}) = -\mathcal{W}(\overline{\varphi_x} - \overline{\varphi_y}).$$

This property is essential and the underlying reason why it is so will become clear soon. In practice, the case when

$$|\overline{\varphi_x} - \overline{\varphi_y}| = \pi$$

is an edge case which rarely happens by chance alone, but when it does (either due to artificially-generated data or a precision error), wrapping in a symmetric interval is necessary to avoid significant errors.

It is easy to show that if φ_x and φ_y are the unwrapped phases corresponding to $\overline{\varphi_x}$ and $\overline{\varphi_y}$, then

$$|\varphi_y - \varphi_x| < \pi \Rightarrow \mathcal{W}(\overline{\varphi_y} - \overline{\varphi_x}) = \varphi_y - \varphi_x.$$

$$|\varphi_y - \varphi_x| > \pi \Rightarrow \mathcal{W}(\overline{\varphi_y} - \overline{\varphi_x}) \neq \varphi_y - \varphi_x.$$

In other words, the estimation $\mathcal{W}(\overline{\varphi_y} - \overline{\varphi_x})$ of the gradient of the unwrapped phase is correct whenever the unwrapped phase difference is strictly smaller than π in absolute value and the estimation is wrong whenever the unwrapped phase difference is strictly greater than π in absolute value. Thus, if the hypothesis on the amplitude of phase differences being relatively small is mostly true, the estimation will be mostly correct. Ignoring edge cases, the proportion of wrongly estimated gradients is the proportion of gradients originally larger than π in absolute value.

The problem is the following: if the true signal is sufficiently smooth, the gradients of the unwrapped phases are all strictly smaller than π in absolute value and the estimation of the gradient on every edge of the graph therefore is correct. In this case, the estimated gradient $\mathcal{W}(\overline{\varphi_y} - \overline{\varphi_x})$ is a true gradient (i.e. the differences of a single function defined on the vertices of the whole graph) and in particular, it is conservative, i.e. the integral over any closed loop of the graph is zero. This means that one can draw a spanning tree in the graph and integrate along it, obtaining the same result (up to an additive constant) regardless of the choice of the tree due to path independence. This result is the estimated unwrapped signal, up to a global constant.

However, it may happen that the profile is very steep or discontinuous, or that the PSs are too far away from one another so that some differences of the unwrapped phases are larger than π. In these cases, the estimated gradient can be wrong (actually, if the true difference is strictly greater

than π, the estimation is obviously necessarily wrong since the wrapping operator sends values into the interval $[-\pi, \pi]$). Since the path independence of the estimated gradient is equivalent to being the gradient of a function on the whole graph, these errors might cause the estimated gradient not to be an actual gradient, leading to a loss of path independence. In this case, integration is not well-defined as it depends on the chosen spanning tree.

If an error is present in an estimation, then it necessarily is an integer multiple of 2π. Therefore, a way to solve this problem is to find the closest path-independent gradient estimation, which can be interpreted as the differences of a global function defined on the vertices of the graph. This closest gradient approximation should be as close as possible to the wrapped gradient for the L_1 norm, with the additional constraint that it should differ from the initial estimation by integer multiples of 2π on each edge.

To do so, one has to minimize the number of edge gradient corrections weighted by their magnitudes, each one being an integer multiple of 2π on each edge, such that the integral of the corrected gradient over every closed graph loop is zero. Once the corrected estimation is found, it remains to integrate over any spanning tree (whose choice does not matter due to the newly gained path independence property) to obtain an estimation of the unwrapped signal up to a global constant.

In practice, the loops considered in the graphs are cycles, defined as the union of distinct undirected edges forming a loop in which each vertex has degree 2. In the graphs we consider, this excludes cycles of length 2. But for a field to be conservative, it also has to sum up to zero when integrated over loops of size 2. Since the corrections added to $\mathcal{W}(\overline{\varphi_y} - \overline{\varphi_x})$ are going to satisfy the antisymmetric property (because of the structure of one of the methods employed, and by definition in the other), summing up to zero on loops of size 2 is equivalent to requiring that $\mathcal{W}$ be antisymmetric. This is the reason why we require $\mathcal{W}$ to be antisymmetric. This article could encompass more general cases of gradient reconstruction, but we shall henceforth and in all cases consider only estimated gradient fields which are antisymmetric, i.e. which satisfy the relationship $\text{grad}(x, y) = -\text{grad}(y, x)$ now taken as granted.

To find these corrections, two different methods can be employed: one relying on a minimum-cost flow interpretation of the problem, which works only on non-redundant Delaunay graphs and when the observed approximate gradient is antisymmetric, but is efficient, and a second one, more general but slower, relying on a linear programming interpretation of the problem, which works in both the redundant and the non-redundant cases. In the latter method, if extra care is taken with loops of size 2, non-antisymmetric estimated gradients could also be handled, although they will not be considered in this article for the sake of simplicity.

2 Related work

Phase unwrapping used to be handled over a regular grid using branch cut methods ([11]). Then L^p-norm methods were developed ([10]). However, it was shown that minimizing the L^2 norm has the tendency to spread errors, whereas minimizing the L^1 norm is more robust to outliers. To minimize this norm, one can employ either Iteratively Reweighted Least Squares (IRLS) [6] or network flow techniques ([2]). Noise being a significant issue in phase unwrapping, causing most errors, it is important to avoid noisy pixels. To do so, one introduces the concept of persistent scatterer or PS, for which candidates of good quality can be found as pixels whose amplitude dispersion, i.e. relative temporal variation in amplitude lies below a certain threshold [8, 9].

The methods described in this paper and implemented in this source code and online demo follow the footsteps of Costantini et al. and come almost entirely from articles [3] and [5] for the minimum-cost flow implementation and the linear programming-based implementation, respectively, with minor tweaks.

The minimum-cost flow (MCF) article [3] describes an implementation using a grid of points. It is easy to see that this implementation can be generalized to any planar graph by exhibiting a 2-basis (i.e. a cycle basis, formally defined in section 3, such that each edge appears in at most two cycles of the basis), which necessarily exists according to Mac Lane's planarity criterion ([17]). In practice, we implement the MCF method to unwrap data using Scipy's Delaunay triangulation, which remains planar even in the event of degeneracy according to its documentation (see: the description of the "neighbors" attribute where it is stated that any elementary triangle has at most one neighbor per edge). The used 2-basis simply is the set of Delaunay simplices (elementary triangles).

The linear programming (LP, not to be confused with norm L_p) version of article [5] also minimizes the L_1 norm ($p = 1$). It is implemented using a redundant Delaunay triangulation with a redundancy parameter r which is a nonnegative integer such that each vertex is linked in the Delaunay triangulation to its k-th nearest neighbors for $1 \leq k \leq r + 1$ (with a single edge, not two edges in different directions). In particular, $r = 0$ corresponds to the case where there is no redundancy, i.e. the graph is the unchanged, default Delaunay graph.

While the MCF method is faster than the LP method, MCF can only be applied to planar graphs, whereas LP can be applied to any graph. In particular, when $r > 0$, the graph is usually non-planar and LP has to be used instead of MCF. In [5], the authors make use of a fundamental cycle basis, which is a basis (defined in section 3) obtained from a spanning tree (see [19]). This property is then used to prove that the solutions (i.e. corrections) are integer multiples of 2π. In other words, the unwrapped solution differs from the wrapped profile by integer multiples of 2π. This is useful because if this were not the case, the solution would be guaranteed to be wrong, at least at those points where the difference is not an integer multiple of 2π. While [19] explains how to build such a basis efficiently, the average cycle length can be large. On the other hand, it is easy to show that any cycle basis preserves the property that the corrections are integer multiples of 2π because the problems using either basis are equivalent. In particular, we use a simple algorithm to find a relatively small (and sometimes minimal) basis for the (possibly redundant) Delaunay triangulations. It can be shown empirically that the smaller average length of the cycles improves computational speed significantly, at least using the PuLP software.

The methods used to solve linear programming problems or minimum-cost flow problems will not be discussed and will be used as is, using open-source software (OR-Tools for MCF and PuLP for LP). More information can be found in [12] or [1]. In particular, well-known yet non-trivial theorems, for instance regarding the property that the LP solutions are integers, will be admitted without proof.

A formula providing a bound on residues (also defined in section 3) depending on the length of the associated cycle can be found in [7]. While not needed in practice, it can be used as a reality check for debugging. Since the article provides no proof, one shall be provided in Appendix A.

3 Method

3.1 Phase unwrapping using linear programming (LP)

A linear program is the following optimization problem:

$$\begin{aligned} &\text{Minimize } c \cdot v \\ &\text{such that } Av = b \text{ and } v \geq 0 \text{ component-wise,} \end{aligned}$$

where c is a real linear form (transposed cost vector or confidence weights, in practice its components are nonnegative), A is a real matrix, b a real vector and v a vector with nonnegative real entries to be determined.

The unwrapping problem outlined in section 1.2, using L_1-norm minimization, can always be made into a linear program, regardless of whether the graph is planar or not. It is the most general method. This advantage is tempered by potentially long computation times depending on the number of points to be unwrapped and the number of edges (which depends essentially on the number of points and the amount of redundancy).

Ensuring the path independence of our estimated gradient is equivalent to checking that integrating it over any cycle yields zero. To satisfy this property, it is also equivalent to requiring this over a subset of cycles called a cycle basis. The concept of basis is the classic one from linear algebra.

Proposition 1. *Assuming our graph G is connected and contains m edges, let $E \cong \mathbb{F}_2^m$, where $\mathbb{F}_2 \coloneqq \mathbb{Z}/2\mathbb{Z}$ denotes the field with two elements, be the associated vector space in which each edge constitutes an element of the canonical basis. The subspace of cycles is the subspace generated by the vectors associated to cycles and a generating, linearly independent subset of these vectors is called a cycle basis. If G contains n vertices, then this subspace has dimension $m - n + 1$.*

This property will be needed to understand how some bases are found in the next section 3.2 and an elementary proof will be provided.

Proof [**Fundamental cycle basis**] Assume a spanning tree T is drawn across G. For every pair of distinct vertices x, y in T such that the edge (x, y) is not in T, associate the cycle comprising edge (x, y) and edges from T. Since T is a tree, it is acyclic and therefore the cycle is unique. Since each such edge (x, y) is contained in a single cycle, this family of cycles is linearly independent. It is easy to convince oneself that these cycles generate the subspace of cycles. Therefore it is a basis. Since G has n vertices, the tree contains $n - 1$ edges, so the cardinality of the basis is $m - n + 1$. This completes the proof. □

Proof [**Linear algebra**] An easier proof relying solely on linear algebra can be obtained by simply noticing that the space $\mathcal{C}$ generated by cycles is exactly the orthogonal of the gradient space ∇ generated by gradients, i.e. $\mathcal{C} = \nabla^{\perp}$ (since a field is a gradient if and only if it is irrotational), and ∇ has dimension $n-1$ because the space of functions on n nodes has dimension n and $\operatorname{Ker} \nabla$ (using the same notation ∇ for the gradient operator, which is uniquely defined in characteristic 2) consists of the subspace of constant functions which has dimension 1 (due to the connectivity property of G, otherwise the kernel would be the subspace of locally constant functions and have dimension c where c denotes the number of connected components of G, leading to the ∇ space having dimension $n - c$ and thus to the cycle space having dimension $m - n + c$ instead of $m - n + 1$).□

Remark 1: the formula for the cardinality of the cycle basis remains true over any field $\mathbb{K}$ instead of $\mathbb{F}_2$ if one uses the alternative, more general definition $\mathcal{C} := \nabla^\perp$ with a cycle basis being a family of vectors constituting a basis of $\mathcal{C}$ such that each vector has nonzero entries only for edges defining a cycle, and such that these entries are equal to ± 1. This new definition generalizes, but is not equivalent to, the one used in Proposition 1 when the characteristic of field $\mathbb{K}$ is different from 2. However, it can be shown that such a cycle basis can be derived from a cycle basis as defined in Proposition 1 in an easy and elementary way. The advantage of using field $\mathbb{F}_2$ is that a cycle is a vector containing only zeros and ones which is unambiguously defined, and considering classes modulo 2 allows for a straightforward understanding of what cycles generate when summed up, e.g. a partition generating its hull. Also, since in characteristic 2 we have $a - b = b - a$, edge orientation does not matter for the gradient field which is also unambiguous. In other fields, if their characteristic is different from 2 (e.g. $\mathbb{K} = \mathbb{R}$), then orientation matters because $1 \neq -1$ and the vectors may contain values other than zero or one, though they can always be found such that their entries are either 0 or ± 1 (see Remark 2).

Remark 2: a cycle basis over $\mathbb{F}_2$ remains linearly independent over $\mathbb{R}$ and can be made into a cycle basis depending on the consistent interpretation of $\overline{1} := 1 \pmod 2$ as either $+1$ or -1 in $\mathbb{R}$, which can be achieved for any undirected graph G by endowing it with a random orientation for each edge. This can be seen by noticing that since the vectors are linearly independent over $\mathbb{F}_2$, the determinant of one of the largest minors of the matrix whose columns are the vectors is nonzero, so it must be an odd integer when the entries are interpreted as reals, which is thus nonzero. Therefore, the matrix has full rank in both fields. However, for the cycle basis over $\mathbb{F}_2$ to be a cycle basis over $\mathbb{R}$, i.e. to be in $\nabla^\perp$, one has to distinguish between ± 1 because an edge can have two different orientations. This can be done by choosing for each cycle any of the two possible orientations (regardless of whether G is planar or not), and consistently identifying $\overline{1}$ as $+1$ if the orientation matches the global orientation originally chosen for G at the given edge, and -1 otherwise. While this might sound trivial, the content of this remark is implicitly used in the following of this section when checking that a field is irrotational over $\mathbb{R}$. Likewise, any basis defined as in Remark 1 over $\mathbb{R}$ remains a cycle basis as defined in Proposition 1 when taken modulo 2, which is obvious because it has the right cardinality and is generating.

Remark 3: One might argue that naively casting a cycle basis defined in Proposition 1 identifying $\overline{0}, \overline{1} \in \mathbb{F}_2$ with $0, 1 \in \mathbb{R}$, respectively, is enough to define a cycle basis over $\mathbb{R}$, rendering the previous remark useless. This is not true, as it would mean that for any graph G it is possible to define a direction on each edge of the graph such that for each cycle of the cycle basis, all of its edges follow the same direction (clockwise or counterclockwise for planar graphs). This is not possible and the simplest example to consider is the undirected planar graph $G = K_4$ from Figure 3 and the cycle basis consisting of the three inner triangles T_1, T_2 and T_3 which are such that T_1 and T_2 share an edge E_{12}, T_1 and T_3 share an edge E_{13}, and T_2 and T_3 share an edge E_{23}, with E_{12}, E_{13} and E_{23} three distinct edges (the inner edges). The three triangles form a cycle basis of G as defined in Proposition 1 because in this case, $m - n + 1 = 6 - 4 + 1 = 3$ and the three triangles are obviously linearly independent over $\mathbb{F}_2$. Picking an orientation (out of the two which are possible, i.e. clockwise or counterclockwise) for T_1 forces T_2 to be oriented in the opposite direction due to shared edge E_{12}. Likewise, T_3 also has to follow the other orientation due to shared edge E_{13}, so that T_2 and T_3 follow the same orientation opposite to that of T_1. Since they share $E_{23} \neq E_{12}, E_{13}$, this is a contradiction.

Remark 4: using the definition of the node-arc incidence matrix of a directed graph provided in section 3.3, one can note that vector space $\mathcal{C}$ is also the space of vectors satisfying Kirchhoff's law of flow conservation at nodes, i.e. divergence is zero at every node. This is interesting because this obser-

vation, along with gradient irrotationality, evoke the well-known formulas $\mathrm{Div} \cdot \mathrm{Rot} = \mathrm{Rot} \cdot \mathrm{Grad} = 0$ and the Helmholtz decomposition theorem in the continuous space $\mathbb{R}^3$.

To define a linear program solving the unwrapping problem of section 1.2, consider the (possibly redundant) Delaunay graph G along with a cycle basis B. If mistakes have been made in the estimations of the gradients, then probably (although not necessarily) there exists at least a cycle such that the integration of the estimated gradients along it yields a nonzero result. This result is called a residue. Since the estimation errors are integer multiples of 2π, so are the residues. The goal is to fix the estimated gradients by adding corrections which we know must be integer multiples of 2π such that integration over each of the cycles yields zero. This is equivalent to requiring that integration over each cycle of a cycle basis yields zero.

The residue for each cycle C is given by

$$\mathrm{Res}_C \coloneqq \sum_{\substack{(x,y)\in C \\ \text{along C}}} f'_{x,y}\,,$$

where we use Costantini's original notation of article [5] with small changes, i.e. $f'_{x,y}$ is the estimated value of the true gradient $f_y - f_x$ where f denotes the original signal to be reconstructed. In our case specific to phase unwrapping, using the notations of section 1.2, we have $f'_{x,y} \coloneqq \mathcal{W}(\overline{\varphi_y} - \overline{\varphi_x})$ whereas f_x denotes the phase at point x before the wrapping process, i.e. $f_x \coloneqq \varphi_x$.

As a reality check, it is easy to show that (cf. Appendix A), if n is the number of edges (or points) in C, then

$$\left|\frac{\mathrm{Res}_C}{2\pi}\right| \leq \begin{cases} \frac{n-1}{2} \text{ if } n \text{ is odd} \\ \frac{n}{2} - 1 \text{ if } n \text{ is even.} \end{cases}$$

In particular, for triangles, the residues lie in $\{0, \pm 2\pi\}$.

We are looking for corrected gradients

$$f_y - f_x = \varphi_y - \varphi_x \coloneqq f'_{x,y} - 2\pi\delta_{x,y} \coloneqq f'^{\mathrm{cor}}_{x,y}$$

with $\delta_{x,y}$ an integer satisfying

$$\delta_{x,y} \coloneqq \delta^+_{x,y} - \delta^-_{x,y}$$

where the $\delta^+ \geq 0$ and $\delta^- \geq 0$ are defined as $\delta^+ \coloneqq \max(\delta, 0)$ and $\delta^- \coloneqq -\min(\delta, 0)$

The corrected gradients must satisfy

$$\sum_{\substack{(x,y)\in C \\ \text{along C}}} f'^{\mathrm{cor}}_{x,y} = 0$$

which is equivalent to requiring

$$\sum_{\substack{(x,y)\in C \\ \text{along C}}} \delta_{x,y} = \sum_{\substack{(x,y)\in C \\ \text{along C}}} \delta^+_{x,y} - \delta^-_{x,y} = \frac{\mathrm{Res}_C}{2\pi} \tag{2}$$

for every cycle $C \in B$, where B is a cycle basis of G. Note that we are implicitly using Remark 2 here by choosing the gradient orientation. This condition can be rewritten as

$$Av = b$$

where $v \geq 0$ is a vector comprising the δ^+ and the δ^- associated to each edge of G (in an arbitrarily fixed direction, the other unknowns are then deduced from the antisymmetry property $\delta_{x,y} = -\delta_{y,x}$ implied by the assumption that the estimated gradient f' is antisymmetric and the fact that the true gradient f'^{cor} naturally satisfies this property). One notes that each of the $m - n + 1$ rows of A corresponds to a cycle of the cycle basis and is full of zeros except at the column indices corresponding to its edges where it is ± 1, depending on its orientation and whether this corresponds to δ^+ or δ^- (so each edge of the cycle has a 1 and a -1 entry corresponding to it and A has $2m$ columns). Of course, A depends on how the δ are stacked in v and one should be mindful of this. For instance, if all the δ^+ are stacked first (in any order) and then all the δ^- (in the same order as the δ^+), then A is of the form $[\tilde{A}, -\tilde{A}]$. Since the residues are integer multiples of 2π, the entries of b are integers. A theorem will show that this property, along with the structure of matrix A, are sufficient to satisfy the requirement that the solutions δ be integers (which were a priori presumed to be reals).

However, we would like to find the nearest true gradient to our estimated gradient in L_1 norm. This is equivalent to requiring the minimization of

$$||v||_1 = \sum_{(x,y)\in\mathcal{E}} |\delta_{x,y}| = \sum_{(x,y)\in\mathcal{E}} \delta^+_{x,y} + \delta^-_{x,y}$$

where $\mathcal{E}$ is the set of edges of G. The equality holds because, by definition, either $\delta^+_{x,y} = 0$ or $\delta^-_{x,y} = 0$

This can be achieved by choosing $c = (1, 1, \ldots, 1)$ and minimizing $c \cdot v = ||v||_1$.

In practice, the components of c are nonnegative and can be interpreted as costs or confidence weights associated to the $\delta^\pm$. A larger value will incentivize the algorithm to prioritize smaller values of the corresponding corrections $\delta^\pm$, which means we have a higher confidence in the corresponding estimated gradient. This is useful when a priori confidence weights can be estimated and produce a custom objective function linear in the $\delta^\pm$, different from the L_1 norm and possibly more accurate. Since the minimization of the objective function is up to a positive multiplicative constant, the weights are values which are relative to each other. Thus, for a generic cost vector c, the objective function becomes

$$c \cdot v \coloneqq \sum_{(x,y)\in\mathcal{E}} c_{x,y}|\delta_{x,y}| = \sum_{(x,y)\in\mathcal{E}} c_{x,y}\delta^+_{x,y} + c_{x,y}\delta^-_{x,y}.$$

Now that the problem has been transformed into a linear program and that finding a cycle basis is outlined in the next section 3.2, it remains to solve for v yielding the minimal corrections to add to our estimated gradient f' to turn it into a true gradient f'^{cor}. This can be done using a linear program modeler such as PuLP, which uses the CBC solver by default. Since B is a basis, every edge has been visited at least once in one direction, thus for each edge (x, y), either $\delta_{x,y}$ or $\delta_{y,x}$ has been computed. If one is missing, then according to the antisymmetric property outlined earlier in the article, it suffices to deduce the other using the relationship $\delta_{x,y} = -\delta_{y,x}$ which we use here by definition, assuming f' to be antisymmetric (in contrast to the other method of section 3.3 in which this relationship is enforced by the flow structure and cannot possibly be adapted to a non-antisymmetric estimated gradient f'). Since this relationship must be satisfied for antisymmetric estimated gradients (the only estimated gradients considered in this article), it is enforced

Algorithm 1: Unwraps data using LP

function *UnwrapLP(Points,Phase,r)*

Input Points: (x,y) coordinates of points $\in \mathbb{R}^2$

Input Phase: Corresponding wrapped phases $\overline{\varphi} \in [-\pi, \pi)$

Input r: Redundancy

Output Unwrapped_phase: Unwrapped phase up to a global constant

$D \leftarrow$ Delaunay($points$) /* Delaunay triangulation */

$Adj \leftarrow$ Adjacency matrix of D

$AdjRed \leftarrow \left(\left(\sum_{k=0}^{r} Adj^{k+1}\right) > 0\right) - (r > 0)\,\mathrm{Id}$ /* Adjacency matrix of the redundant Delaunay graph */

$Edges \leftarrow AdjRed > 0$ /* Edges of the redundant Delaunay graph */

$f'_{x,y} \leftarrow \mathcal{W}\left(\overline{\varphi_y} - \overline{\varphi_x}\right), \forall (x,y) \in Edges$ /* Wrapped gradients */

$B \leftarrow$ GetCycleBasis($D, Adj, AdjRed$) /* Algo 2 or Algo 4 */

for C **in** B **do**

  Res $_C \leftarrow \sum_{\substack{(x,y)\in C \\ \text{along C}}} f'_{x,y}$

  Add constraint $\sum_C \delta_{x,y} = \frac{\mathrm{Res}_C}{2\pi}$ for (x,y) edge along C in a consistent direction

  and with $\delta_{x,y} \coloneqq \delta^+_{x,y} - \delta^-_{x,y}; \delta^+_{x,y} \geq 0, \delta^-_{x,y} \geq 0$

Set objective function: $\sum_{(x,y)\in\mathcal{E}} \delta^+_{x,y} + \delta^-_{x,y}$ /* Set objective function */

$\delta^{\pm} \leftarrow$ Solve linear program, e.g. using PuLP

$f'^{\mathrm{cor}} \leftarrow f' - 2\pi\delta$ /* Corrected gradients */

Unwrapped_phase $\leftarrow$ Integrate f'^{cor} over any spanning tree, start from a constant, e.g. 0

return Unwrapped_phase

in the code. Otherwise, cycles of size 2 would need to be taken into account. Since f'^{cor} is a true gradient, it is in particular conservative and integrating it over any spanning tree T of G yields the same estimation of the unwrapped phase profile, which is thus well-defined up to a global constant.

If the cycle basis is totally unimodular (e.g. a fundamental basis obtained from a spanning tree [18]), i.e. if the corresponding matrix of constraints A is totally unimodular (meaning that every square submatrix of A has determinant 0 or ± 1), the δ are guaranteed to be integers ([12], p.738, Th.1) even if the variables δ are chosen to be continuous, i.e. varying in $\mathbb{R}$. This is recommended with PuLP as it runs significantly faster than if we require that the variables be in $\mathbb{Z}$ from the outset. This is convenient since we know that the real corrections must be integer multiples of 2π. However, our cycle basis is not always totally unimodular. This is no cause for concern, though, due to the following reasoning: suppose we choose a fundamental cycle basis B, i.e. a cycle basis obtained from a spanning tree, which can be done systematically by collecting the unique cycle corresponding to each edge of the graph not in the tree and the only path in the tree connecting its extremities like in the proof of Proposition 1 (see section 3.5.3 and Figure 10 for a visual example). Then we have to minimize $c \cdot v$ with the conditions $Av = b$, $v \geq 0$ and A totally unimodular. Since any cycle basis B' spans the same cycle space $\mathcal{C}$, the matrix of constraints A' corresponding to this alternative basis satisfies

$$\mathrm{Ker}\, A' = \mathcal{C}^{\perp} = \mathrm{Ker}\, A$$

This is equivalent to requiring the existence of an invertible matrix P satisfying

$$A' = PA$$

Multiplying by P the constraints in the original problem, one gets the equivalent problem

$$Av = b \Leftrightarrow PAv = A'v = b' = Pb$$

where A' is not necessarily totally unimodular.

Because the problems are equivalent (since P is invertible), the solutions (which are not necessarily unique) are the same. The well-known theorem ([12], p.738, Th.1) states that the solutions are integers for any vector b with integer entries and any vector c if A is totally unimodular. Since this is the case for the basis B, the solutions are integers. Since the problem is equivalent using basis B', the solutions remain integers even if A' lacks the total unimodularity property. Therefore the solutions δ are integers no matter which cycle basis we choose.

A last remark concluding this section should be made concerning the potentially degenerate Delaunay triangulation (which is often encountered in practice due to some degree of regularity in the data points, e.g. grid-like). One simply has to note that none of the reasoning above assumes the triangulation to be non-degenerate. The only property which has been mentioned (but not used yet) which could be affected is the default Delaunay triangles constituting a 2-basis, should an edge be represented more than twice in the cycle basis. However, this property is not needed for this section, and even if it were the case, Scipy's Delaunay implementation eschews this issue using option QJ (this feature will be a key element for the next section concerning MCF, without which it will be shown that the technique cannot work properly). Therefore the above reasoning still holds even for degenerate Delaunay. The reasoning of the next section will also hold as long as the set of default Delaunay triangles remains a 2-basis, which is the case when one uses Scipy's Delaunay implementation with option QJ as outlined earlier.

The pseudocode in Algorithm 1 provides a summary of this section.

One final note concerning the code and the computation of the residues: according to section 1.2, we know that

$$f'_{x,y} \coloneqq \mathcal{W}(\overline{\varphi_y} - \overline{\varphi_x}) = \overline{\varphi_y} - \overline{\varphi_x} + 2\pi a_{x,y}$$

where $a_{x,y} \in \{0, \pm 1\}$ is the integer ambiguity that wraps the phase according to Equation 1. We also know that summing over a cycle, the $\overline{\varphi_y} - \overline{\varphi_x}$ will vanish, so instead we can only compute the ambiguities and convert a numerical computation into an essentially infinite-precision, error-free computational algebra problem giving the right-hand side term of the constraint of Equation 2 according to

$$\frac{\mathrm{Res}_C}{2\pi} = \sum_{\substack{(x,y)\in C \\ \text{along C}}} a_{x,y}.$$

This is what the demo code does in practice.

3.2 Finding a cycle basis

While spanning trees are a convenient way of finding a basis in a general graph, it has the flaw that the obtained basis sometimes contains relatively large cycles. This entails time-consuming computations. Since we are dealing with a (possibly redundant) Delaunay graph, there are more effective ways to find bases with a small average number of edges per cycle and a small upper bound on it. In two of the most common scenarios, one can produce minimum cycle bases, namely bases with the least number of edges possible. We shall only keep in mind that the sought basis has cardinality $m-n+1$.

The easiest case is the case when G is a Delaunay triangulation with no redundancy. It is easy to see that the family of elementary triangles generated by Delaunay forms a basis. Indeed, assume the vectors associated to the triangles are $\{v_i\}_i$ and assume that $\sum \lambda_i v_i = 0$ with $\lambda_i \in \mathbb{Z}/2\mathbb{Z}$ for every i. Since an edge on the boundary belongs to a single triangle, we have $\lambda_j = 0$ for every j corresponding to the triangles with at least an edge on the boundary of G. Removing these triangles, we get another graph G'. Iterating this technique by exhaustion, one gets $\lambda_i = 0$ for every i, therefore the family is linearly independent. Another way to see this is that if the λ_i are not all identically zero, the sum corresponds to the hull of the triangles, which is a non-empty union of cycles. Indeed, it is easy to see that each cycle of a Delaunay triangulation is the sum of the triangles it encloses. Thus, these triangles form a basis. Since every cycle of the basis has length 3 and there are no cycles of length 2 or self-loops (loops of size 1), it is a minimum cycle basis, i.e. a basis with the least total number of edges.

The second easiest case is when G is a Delaunay triangulation with redundancy 1. This means that every vertex is linked to its neighbors in the default Delaunay graph and additional edges are added to connect it to its next-nearest neighbors. In this case, we can start with the family of triangles of the default Delaunay triangulation. For each additional edge linking a vertex to a next-nearest neighbor, the basis cardinality $m - n + 1$ grows by one unit, so a new linearly independent cycle has to be found. This cycle can be obtained as the union of the new edge with two edges forming a path between the extremities of the new edge in the default Delaunay triangulation (ignoring the new edges). This path exists by definition of the new edge as being the result of redundancy 1. Since the new edge belongs to a single cycle, it is linearly independent of the others. Proceeding this way for all the new edges, one gets a basis of triangles. This basis is thus a minimum cycle basis, because there are no self-loops and no loops of size 2 (redundant edges are only counted once).

In the general case, when the redundancy satisfies $r > 1$, one proceeds in the same way as above. One starts with the family of default Delaunay triangles, and for each new edge, one adds a cycle containing this edge and a path between the extremities in default Delaunay (or in the union of default Delaunay with the additional edges added before the edge considered, but not the edges which will be added in the future). This will form a relatively small basis (each cycle having length $\leq r + 2$). To find the path, an easy way is to create two breadth-first clusters at each extremity and grow them alternatively and iteratively until they meet. A breadth-first cluster at a node is simply the subgraph containing the node plus its i-th nearest neighbors for $1 \leq i \leq j$ starting with $j = 1$,

Algorithm 2: Obtain a small cycle basis

```
1 function GetSmallBasis(D,Adj,AdjRed)
      Input D: Scipy's Delaunay triangulation
      Input Adj: Adjacency matrix of the Delaunay triangulation
      Input AdjRed: Adjacency matrix of the redundant Delaunay triangulation
      Output B: A small cycle basis of redundant Delaunay
2     B ← D.simplices                          /* Elementary triangles */
3     R ← (AdjRed − Adj ≠ 0)                   /* Redundant edges */
4     for E = (x, y) in R do
5         C = GetShortestPath(D, x, y)
6         B ← B ∪ {C}
7     return B
```

and growing it means incrementing j by one unit. Growing two clusters is better than growing only one because it divides the distance to be traveled by 2, which is significant as breadth-first clusters grow exponentially with the distance, though since r is generally small, this process is relatively quick (a path is obtained after at most $r + 1$ iterations in both cases but in the case involving two clusters, the number of visited nodes is reduced).

Visual representations are provided in Figure 1.

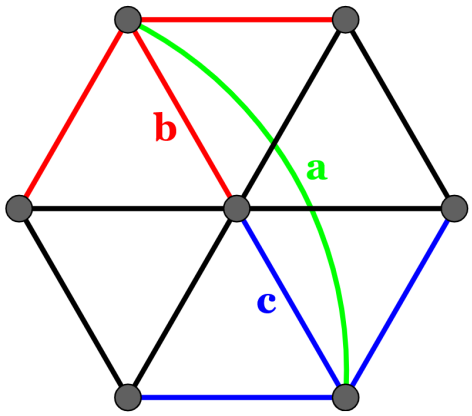
There remains a caveat, however: while it can be shown that the cycle basis obtained from a spanning tree, called a fundamental cycle basis, is always a totally unimodular basis (which means that the matrix A of constraints of the previous section 3.1 is totally unimodular, i.e. every submatrix of A has determinant 0 or ± 1), this is not the case for the bases obtained using the above technique. There is an exception though, when there is no redundancy, as the elementary triangles of the default Delaunay triangulation form a 2-basis, a property which can be shown to entail total unimodularity: in this case, it suffices to use the row partition lemma 2 with $I_1 = \{\text{All the rows}\}$ and $I_2 = \emptyset$ on A.

Algorithm 3: Obtain a shortest path between two points in a connected graph

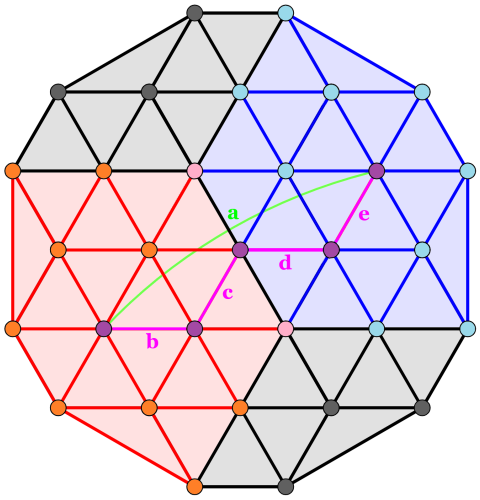
```
1  function GetShortestPath(G,x,y)
       Input G: Any connected graph
       Input x,y: Extremities of the shortest path to be found in V_G
       Output P: A shortest path from x to y in G
2      Cluster_x ← {x}
3      Cluster_y ← {y}
4      Intersection_test ← 0
5      Counter ← 0
6      while Intersection_test == 0 do
7          if Cluster_x ∩ Cluster_y == ∅ then
8              if Counter mod 2 == 0 then
9                  Grow Cluster_x by including its nearest neighbors in G
10                 Store information on predecessors of nodes in Cluster_x
11             else
12                 Grow Cluster_y by including its nearest neighbors in G
13                 Store information on predecessors of nodes in Cluster_y
14             Counter ← Counter +1
15         else
16             Intersection_test ← 1
17             Store an intersection node V ∈ Cluster_x ∩ Cluster_y
18     P_1 ← Reconstruct path from x to V using information on predecessors in Cluster_x
19     P_2 ← Reconstruct path from V to y using information on predecessors in Cluster_y
20     P ← P_1 ∪ P_2
21     return P
```

Algorithm 4: Obtain a fundamental cycle basis following the guidelines of [19]

```
1  function GetFundamentalBasis(V,E)
       Input V_G: Vertices of the graph G
       Input E_G: Edges of the graph G
       Output B: A fundamental cycle basis of any graph G
2      V_T ← {v}  /* Root of the spanning tree to be grown chosen at random in V */
3      E_T ← ∅                          /* Edges of the spanning tree to be grown */
4      X ← V_G                                  /* Vertices not yet examined */
5      while V_T ∩ X ≠ ∅ do
6          z ← random element from V_T ∩ X
7          for w such that (z, w) ∈ E_G do
8              if w ∈ V_T then
9                  C = GetShortestPath(T, z, w)
10                 B ← B ∪ {C}
11             else
12                 V_T ← V_T ∪ {w}
13                 E_T ← E_T ∪ {(z, w)}
14             E_G ← E_G \ {(z, w)}
15         X ← X \ {z}
16     return B
```



(a) Simple illustration of the creation of a cycle involving a redundant arc a with redundancy 1 in green in a Delaunay triangulation of 7 vertices. The two breadth-first clusters, in red and blue, intersect after 2 iterations (one for each cluster) at the node in the center. The new cycle corresponding to a is thus cycle (a, b, c).



(b) Simple illustration of the creation of a cycle involving a redundant arc a with redundancy 3 in green in a larger Delaunay triangulation. The two breadth-first clusters, in red and blue, intersect after 4 iterations (two for each cluster) and an instance of a possible path completing the cycle is shown in pink. The new cycle corresponding to a is thus cycle (a, b, c, d, e).

Figure 1: Examples of cycle creation from a redundant arc.

Proposition 2 (Row partition lemma ([12], p.738, Th.2))**.** *A matrix A whose entries belong to $\{0, \pm 1\}$ is totally unimodular if each column contains no more than two nonzero entries and if its set of rows can be partitioned into two sets I_1 and I_2 such that if a column has two entries of the same sign, the corresponding rows belong to different sets of the partition, and if a column has two entries of different signs, the corresponding rows belong to the same set of the partition.*

In the general case, counter-examples can be found. It has been shown nonetheless in the previous section that since both problems (the one using the tree basis and the one using the small basis) are equivalent, a well-known (yet non-trivial) theorem ensures that the corrections remain integer multiples of 2π, which is important since we know that this property must be satisfied by the real corrections. The only requirement is that at least one totally unimodular cycle basis exists, and since one can always build a fundamental cycle basis using any spanning tree, this condition is always satisfied.

In some cases when one does not use Delaunay, a general fundamental basis might need to be exhibited. In these cases, one simply follows the guidelines given in [19], where a spanning tree is grown alongside the fundamental cycle basis.

The three algorithms 2, 3 and 4 in pseudocode summarize this section.

3.3 Phase unwrapping using minimum-cost flow (MCF)

The minimum-cost flow technique consists in interpreting the δ of section 3.1 as flows on the edges of a network dual to Delaunay and such that for each vertex, the supply (i.e. flow out minus flow in) equals an integer associated to a residue from a corresponding Delaunay triangle. The objective function is the same as the one in the previous section and the minimum-cost flow problem consists in finding a flow which minimizes this function. The advantage of MCF over LP is that it is faster, the downside is that MCF only applies to planar graphs and the δ are necessarily antisymmetric.

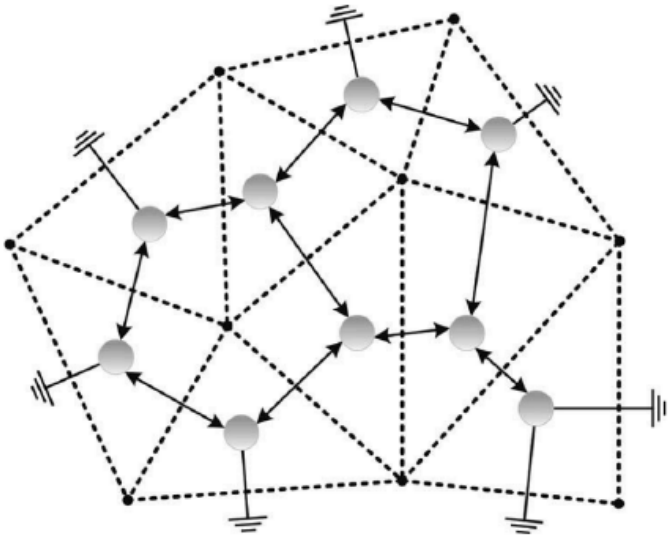
This technique can be applied to any graph for which a 2-basis can be exhibited. According to Mac Lane's planarity criterion, this is equivalent to requiring that the graph be planar. In practice, since when $r > 0$ the redundant Delaunay graph is almost always non-planar, we shall focus only on the case $r = 0$. In this case (default Delaunay), the elementary triangles form a cycle basis which is obviously a 2-basis when the triangulation is non-degenerate (by definition of a polygon triangulation). Using Scipy's implementation of Delaunay, this cycle basis remains a 2-basis even when degeneracy occurs, as long as one uses option QJ (a slight joggling of the input points with a reasonable probability density (e.g. locally uniform or normal) before triangulating ensures a planar Delaunay triangulation almost surely).

Because of this property, each edge of the Delaunay triangulation is involved in exactly two triangles or faces (possibly degenerate), except for the edges of the convex hull which are involved in a single triangle. However, counting the exterior as a face (which in this setting is usually called the Earth), every edge appears at the boundary of exactly two faces. Better yet, an edge endowed with a direction appears in a single triangle counterclockwise (counting the exterior face called the Earth when no such triangle exists and endowing degenerated triangles with the handedness provided by Delaunay's joggling option QJ which removes degeneracy almost surely). This means that each directed edge corresponds to a triangle (or the Earth), and each triangle corresponds to three directed edges (its edges with the counterclockwise direction).

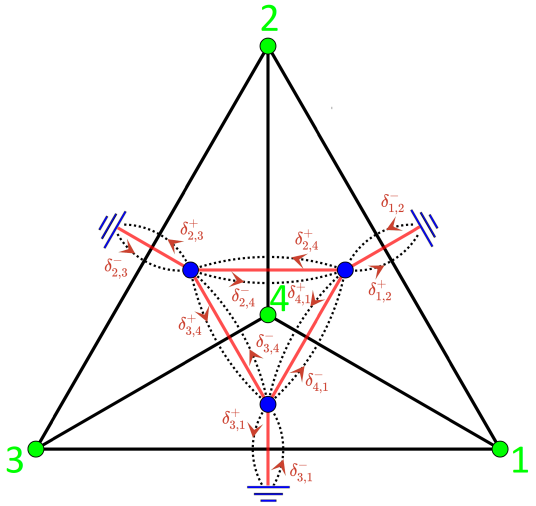
Let us now define the dual network. Each elementary triangle corresponds to a vertex, includ-

ing the exterior face which counts as a vertex called the Earth. For a given triangle, each of its three edges in the Delaunay triangulation appears in a single neighboring face. One simply connects the nodes corresponding to the triangle and the neighboring face by two directed edges, one going outside the triangle and one going inside it.

Instances of visual representations of triangulations along with their dual graphs are provided in Figure 2.



(a) Illustration of the dual graph (solid lines) of a triangulation (dashed lines). The double arrows, along with the line segments to the Earth, denote two arcs in opposite directions. This figure was borrowed from article [24].



(b) Basic but more faithful illustration of a Delaunay triangulation (green vertices and solid black line segments) along with its dual graph (blue vertices and solid red line segments) with bidirectional edges (dashed) and a possible configuration of the $\delta^{\pm}$ (red arrows). Note that for each edge (i,j), only one of $\delta_{i,j}$ or $\delta_{j,i}$ appears, so one has to deduce the other by relying on the antisymmetric property $\delta_{i,j} = -\delta_{j,i}$.

Figure 2: Typical dual setting for implementing MCF. The Earth (outer face), despite being represented multiple times, is actually a single node.

Now that the dual graph is defined, one simply notes that, using the notations of the previous sections, we have for each triangle C

$$\sum_{\substack{(x,y)\in C\\ \text{counterclockwise}}} \delta_{x,y} = \sum_{\substack{(x,y)\in C\\ \text{counterclockwise}}} \delta^+_{x,y} - \delta^-_{x,y} = \frac{\text{Res}_C}{2\pi}.$$

Therefore it suffices to assign to each node an integer supply equal to the residue of the triangle divided by 2π and for each directed edge (x,y) to define the flow going outside the corresponding triangle as $\delta^+_{x,y}$ and the flow going inside as $\delta^-_{x,y}$. Indeed, in this case, it is trivial to check that for any given flow, integrating the corrected gradient f'^{cor} over the triangle yields the residue of the triangle minus 2π times the supply of the corresponding dual node, which sum up to zero. In other words, integrating f'^{cor} over the cycle basis yields zero. Note that with this flow definition, $\delta_{x,y} = -\delta_{y,x}$ is automatically verified, so if the wrong, estimated gradient is antisymmetric, so is the corrected gradient and it suffices that it sum up to zero over triangles for it to be a true gradient. Otherwise, the method will not work, as the 'corrected' gradient will still leave nonzero residues on some loop of size 2, so it will not be conservative, i.e. a true gradient. One ought not to forget to account

Algorithm 5: Unwraps data using MCF

function *UnwrapMCF(Points,Phase)*
  Input Points: (x,y) coordinates of points $\in \mathbb{R}^2$
  Input Phase: Corresponding wrapped phases $\in [-\pi, \pi)$
  Output Unwrapped_phase: Unwrapped phase up to a global constant
  $D \leftarrow$ Delaunay$(points)$ /* Triangles of the Delaunay triangulation */
  $G \leftarrow$ Dual graph of the Delaunay triangulation
  $Adj \leftarrow$ Adjacency matrix of D
  $Edges \leftarrow Adj > 0$ /* Edges of the Delaunay graph */
  $f'_{x,y} \leftarrow \mathcal{W}\left(\overline{\varphi_y} - \overline{\varphi_x}\right), \forall (x,y) \in Edges$ /* Wrapped gradients */
  Supply $\leftarrow \emptyset$
  for C **in** D **do**
    Compute residue Res_C from integrating $f'_{x,y}$ over C counterclockwise
    Supply $\leftarrow$ Supply $\cup \{\frac{\mathrm{Res}_C}{2\pi}\}$
  $\mathrm{Res}_{\mathrm{Earth}} \leftarrow -\sum_{C \in D} \mathrm{Res}_C$
  Supply $\leftarrow$ Supply $\cup \{\frac{\mathrm{Res}_{\mathrm{Earth}}}{2\pi}\}$
  Set appropriate capacities (usually $+\infty$)
  $c \leftarrow (1, 1, \ldots, 1)$ /* Set uniform objective or cost function */
  $\delta^{\pm} \leftarrow$ Solve minimum-cost flow problem over G, e.g. using OR-Tools
  $f'^{\mathrm{cor}} \leftarrow f' - 2\pi\delta$ /* Corrected gradients */
  Unwrapped_phase $\leftarrow$ Integrate f'^{cor} over any spanning tree, start from a constant, e.g. 0
  return Unwrapped_phase

for the Earth node, or the face exterior to the convex hull of the graph. It is easy to show, either using a graph-theoretical divergence theorem (no flow leaks outside the graph so the sum of the divergences, i.e. supplies, is zero) or a graph-theoretical Stokes theorem, that its supply is equal to $\mathrm{Res}_{\mathrm{Earth}} = -\sum_C \mathrm{Res}_C$ for C running over all the Delaunay triangles.

If v is a vector comprising the $\delta^{\pm}$, then finding a flow is equivalent to solving

$$Av = b,$$

where A is a node-arc incidence matrix of the dual graph and b a vector with integer entries corresponding to the residues. A node-arc incidence matrix of a directed graph G with n nodes and m edges is an $n \times m$ matrix where each column corresponds to a directed edge and is full of zeros except at the line index corresponding to the starting node which is 1 and at the line index corresponding to the end node which is -1. The matrix A is thus totally unimodular, as one can use the row partition lemma 2 with $I_1 = \{\text{All the rows}\}$ and $I_2 = \emptyset$. The minimum-cost flow problem consists in exhibiting a vector $v \geq 0$ satisfying the above condition and minimizing

$$c \cdot v.$$

So MCF is just a particular case of LP for which fast solvers exist. In particular, due to the theorem invoked in the previous section, the solutions δ remain integers. In our case, we usually pick $c = (1, 1, \ldots, 1)$ just like in section 3.1. Then one can use an MCF solver such as the one provided by OR-Tools to find the δ and infer the corrected gradient f'^{cor} used to unwrap the data by integrating over any spanning tree. It is easy to see that this definition of the $\delta^{\pm}$ along with this objective function implies, for any edge (x, y), $\delta^+_{x,y} = 0$ or $\delta^-_{x,y} = 0$ (because otherwise we could decrease

each of the $\delta^{\pm}_{x,y}$ by one unit and get a new feasible flow with a cost reduced by two units). Therefore, we recover implicitly the old definition of $\delta \coloneqq \delta^{+} - \delta^{-}$ with $\delta^{+} = \max(\delta, 0)$ and $\delta^{-} = -\min(\delta, 0)$.

The general theory of minimum-cost flow problems allows one to upper-bound the flows $\delta^{\pm}$ using so-called capacities. However, in practice, most of the time, the capacities will be ignored and set to infinity. Nevertheless, if a bound β is known in absolute value for the true gradients, then since the wrapped gradients lie in $[-\pi, \pi]$, the correction to be added to the estimated gradients in integer units of 2π is bounded by $\lfloor \frac{\beta+\pi}{2\pi} \rfloor$. Therefore one can set uniform capacities on all edges, yielding $0 \leq \delta^{\pm} \leq \lfloor \frac{\beta+\pi}{2\pi} \rfloor$.

The pseudocode described in Algorithm 5 summarizes this section.

3.4 When the δ are not integer multiples of 2π: the generic method

It might arise that in some situations, we estimate the gradient of a function other than phases using a regression associated with a model, such as the estimation of the gradients of the deformation velocity and the residual topographic error in the PS technique [8]. In this case, the aforementioned δ in the previous sections are not integer multiples of 2π anymore, i.e. the approximated gradient does not match the actual gradient modulo 2π point-wise. In other words, the optimal δ regarding LP or MCF are not necessarily integers and the approximate gradient f' doesn't come from a wrapping operator anymore. However, the LP method remains valid without changing anything other than the approximate gradient f' and the formula of the bound of the gradients which doesn't make sense anymore. The only exception concerns the factors involving 2π, which are such that 2π can be set to 1 to do away with unneeded complexity (though leaving the methods as is would still function). The relationship between the observed gradient f' and the corrected gradient f'^{cor} is now given by

$$f_y - f_x = f'_{x,y} - \delta_{x,y} \coloneqq f'^{\mathrm{cor}}_{x,y}. \tag{3}$$

The result is the gradient closest in L_1 norm to the approximate gradient, with real point-wise differences $\delta_{x,y}$, not necessarily integer multiples of 2π.

When it comes to MCF, since OR-Tools supports only integers, the method cannot be used as is and would return spurious results. However, since MCF was used in the case modulo 2π by storing the integers such that $\delta \in 2\pi\mathbb{Z}$, thereby approximating a true gradient by adding integer multiples of 2π, it should be possible to use the same reasoning by replacing 2π by a very small value $\epsilon > 0$. To ensure the solutions δ remain integers, the gradients on each edge are divided by ϵ and rounded to the nearest integer as shown in Equation 4, where $\lfloor . \rceil$ denotes the rounding operation.

$$f_y - f_x = \left\lfloor \frac{f'_{x,y}}{\epsilon} \right\rceil \epsilon - \delta_{x,y}\,\epsilon \coloneqq f'^{\mathrm{cor}}_{x,y}. \tag{4}$$

If ϵ is small enough, the rounded-out part is relatively small and does not significantly affect the end result. Of course, the smaller the gradients in absolute value, the smaller ϵ should be. Setting ϵ exceptionally small leads to intensive calculations and long computation times, but setting it too large can lead to imprecision, which can add up upon integration. A good trade-off for most intents and purposes is to set $\epsilon = 10^{-3}$, but this could depend on the magnitudes of the gradients. One should also exercise caution by setting extremely high capacities, such as 10^9 to be safe, because None isn't supported and a very small ϵ can lead to very large residues and flows (typically on the order of $\frac{1}{\epsilon}$).

Finally, note that only antisymmetric gradient estimates are supported here as well. This means that we make the reasonable assumption that the estimated gradient from x to y is minus the estimated gradient from y to x and that the rounding operator is antisymmetric, e.g. by rounding up positive half-integers and down negative half-integers.

3.5 Instructive elementary examples

Before proceeding to the experiments, which were conducted on graphs containing several thousands of points, it might be instructive for beginners to better understand the concepts and machinery above using examples based on small graphs with at most a few dozen points. This section contains interesting mathematical curiosities which are nonetheless either very well-known or problems so simple that they can be solved by hand, but can shed some light for the inexperienced. The seasoned reader might want to skip this section entirely.

3.5.1 Planar vs. non-planar graphs: towards another characterization

The two most common classes of graphs are the complete graphs and the complete bipartite graphs.

To allow for intuitive proofs, one might cite Fáry's theorem 1 which states that a graph is planar if and only if the edges can be drawn without crossings such that they form straight line segments. Therefore, in the following, we shall henceforth only consider planar configurations where edges are straight line segments.

Theorem 1 (Fáry's theorem [22])**.** *A graph is planar if and only if it can be drawn in the plane such that its edges do not cross and are straight line segments.*

The complete graph with n vertices, denoted by K_n, is the graph containing n vertices and the maximum number of edges, i.e. every pair of vertices is linked by an edge, so the number m of edges is $\binom{n}{2}$. One can show that K_n is planar if and only if $0 \leq n \leq 4$. The first few complete graphs are shown in Figure 3 and the planarity of K_n for $0 \leq n \leq 4$ is obvious.

An elementary proof that K_5 is non-planar is the following: assuming only straight edges, one first draws two vertices $1, 2$ at any two different positions in the 2D Euclidean plane. Then one adds the third vertex 3 anywhere on the plane which does not lie on the line containing the first two vertices $1, 2$, otherwise the graph contains edges which would cross each other. This creates a non-degenerate triangle $(1, 2, 3)$, i.e. a triangle with a non-empty interior. Drawing the fourth vertex while avoiding an intersection with the triangle necessarily leads to a configuration where a triangle is partitioned into three others (see Figure 4). But then, it becomes obvious any fifth vertex will induce a crossing. Therefore, K_5 is necessarily non-planar.

An obvious corollary is that K_n is non-planar for $n \geq 5$, because it contains K_5 as a subgraph.

The complete bipartite graph with vertices n_1, n_2, denoted by K_{n_1,n_2}, is the graph containing $n_1 + n_2$ vertices with two sets of vertices forming a partition, set s_1 containing n_1 vertices and set s_2 containing n_2 vertices. Each vertex of s_1 is linked to all the vertices of s_2 (and as a result, each vertex of s_2 is linked to all the vertices of s_1). Its number of edges is thus $n_1 n_2$. The first few complete bipartite

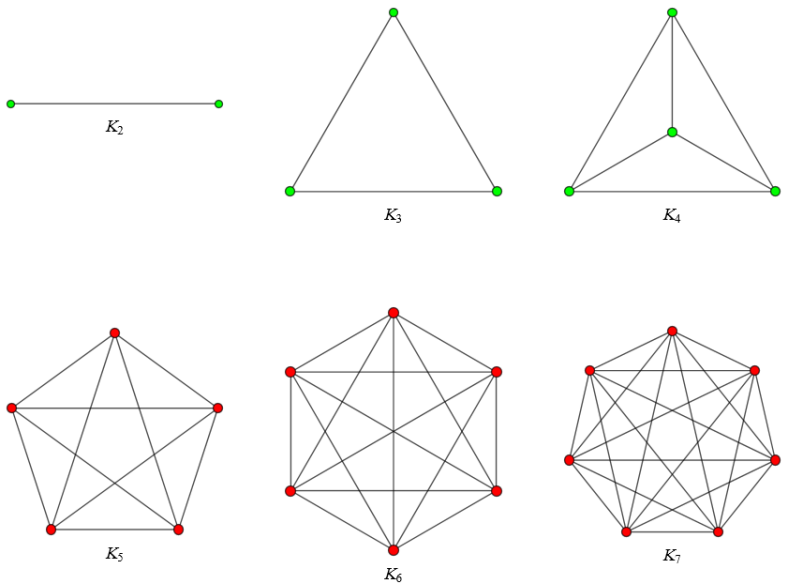


Figure 3: A list of complete graphs K_n for $2 \leq n \leq 7$. Planar graphs are drawn with green vertices while non-planar ones are drawn with red vertices.

graphs are shown in Figure 5.

It is clear that $K_{1,1}$, $K_{1,2}$ and $K_{1,3}$ are planar from Figure 5. It is also clear that $K_{2,2}$ and $K_{2,3}$ are planar despite the crossings in Figure 5 as one can move the bottom left vertex of the corresponding graphs far to the right, leading to a planar graph configuration in both cases. Using the same reasoning, one shows that $K_{1,n}$ and $K_{2,n}$ are planar for any $n \in \mathbb{N}$.

However, $K_{3,3}$ is non-planar. An elementary proof is the following: assuming straight edges, place any two distinct vertices $1, 2$ belonging to s_1. Then place vertex 3 belonging to s_2. It can lie anywhere on the plane except on the line containing vertices 1 and 2, thereby forming triangle $(1, 2, 3)$ and yielding Figure 6. Place vertex 4 also belonging to s_2. If it lies within the light-green region of the topmost illustration of Figure 6, then this leads to two triangles $(1, 2, 3)$ and $(1, 2, 4)$ (illustration on the left in the middle), one containing the other, which then necessarily leads to an impossibility while drawing the last vertex in s_1 (last row of illustrations). Otherwise, we get the middle right illustration, and upon placing the fifth vertex in s_2, we get a special case of the illustration of the middle left, which was shown to lead to an impossibility. Therefore, $K_{3,3}$ is non-planar.

An obvious corollary is that K_{n_1,n_2} is non-planar for $n_1, n_2 \geq 3$, because it contains $K_{3,3}$ as a subgraph.

Now that the planarity of two of the most common classes of graphs has been characterized, a mathematical curiosity is worth mentioning: in the two classes, the first two non-planar graphs encountered were K_5 and $K_{3,3}$. It happens that they occur in a characterization of the planar graphs: these two graphs constitute, in a way, an obstruction to planarity. They are sometimes called "forbidden graphs".

To introduce the theorem, one requires the definitions of graph isomorphisms, graph subdivisions

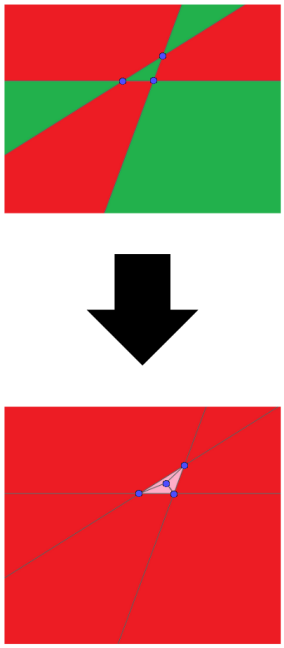


Figure 4: The first figure shows the first three vertices and in green are the regions where the fourth vertex may be drawn. The second figure shows an example of a drawing with four vertices, leading to the impossibility of drawing a fifth vertex (in red are the regions which would induce a crossing with the larger triangle and in pink are the regions which would induce a crossing inside the triangle).

and graph homeomorphisms.

Definition 1 (Graph isomorphism)**.** *Two graphs G and H are said to be isomorphic if there exists an edge-preserving permutation σ of the vertices of G such that $\sigma(G) = H$. Edge preservation means edge (i, j) exists in G if and only if $(\sigma(i), \sigma(j))$ exists in H.*

Definition 2 (Graph subdivision)**.** *A subdivision of G is a graph H obtained from G by adding an arbitrary number (possibly 0) of vertices $ij_1, ij_2, \ldots, ij_{n_{ij}}$ to its edges (i, j) such that (i, j) is replaced by edges $(i, ij_1), (ij_1, ij_2), \ldots, (ij_{n_{ij}}, j)$ and the new vertices ij_k are not involved in any other edges than these ones.*

Definition 3 (Graph homeomorphism)**.** *Two graphs G and H are homeomorphic if there exist a subdivision G' of G and a subdivision H' of H such that G' and H' are isomorphic.*

The exact statement of the aforementioned planarity characterization theorem based on forbidden subgraphs is the following:

Theorem 2 (Kuratowski's theorem [15])**.** *A finite graph is planar if and only if it does not contain any subgraphs homeomorphic to K_5 or $K_{3,3}$.*

Corollary 1. *A finite graph is non-planar if it contains either K_5 or $K_{3,3}$ as one of its subgraphs.*

Caution: the converse of Corollary 1 is false because subgraphs homeomorphic to K_5 or $K_{3,3}$, also called Kuratowski subgraphs, are not restricted to K_5 or $K_{3,3}$.

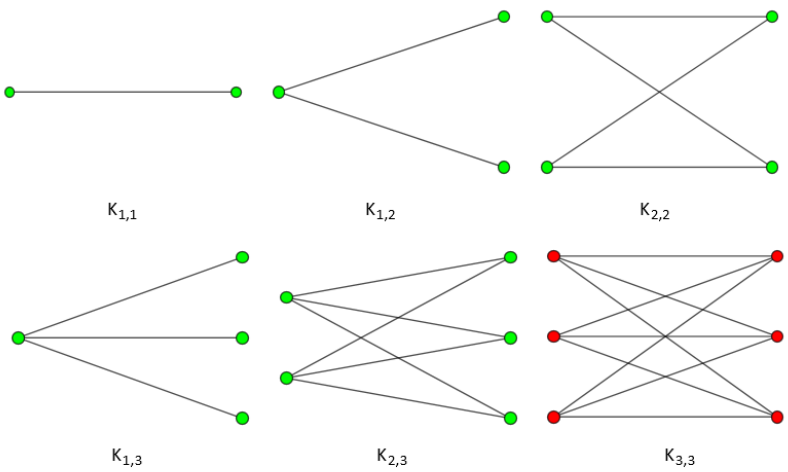


Figure 5: A list of complete bipartite graphs K_{n_1,n_2} for $1 \leq n_i \leq 3$ and $n_1 \leq n_2$. Planar graphs are drawn with green vertices while non-planar ones are drawn with red vertices. The graphs are drawn using a canonical form, so some graphs can exhibit crossing edges despite their being planar.

3.5.2 Gradient correction on elementary examples

It can be insightful to present some very elementary gradient correction examples which can be solved without a computer to better understand the machinery involved in the previous sections. As before, only connected graphs are considered, since for disconnected graphs one can simply process each connected component independently.

The first class of elementary graphs on which gradient correction is trivial might be worth mentioning for beginners not yet accustomed to the theory. It concerns connected graphs which are acyclic, i.e. which contain no cycles (see Figure 8 for a few examples of cycles). These graphs are trees (see Figure 7).

The theory of the previous sections still applies as is, and if the number of vertices is n then the number of edges is $m = n - 1$ and as a sanity check, one notes that the formula still holds: the cycle basis has cardinality $m - n + 1 = 0$, i.e. the space generated by cycles is the zero vector space, as expected. This means its orthogonal, the gradient space, is the whole space. As a result, any gradient estimates on its edges is a gradient and there is nothing to do. The solution is $\delta^{\pm} = 0$ and the gradient is the differences of the class of functions obtained by integrating on the tree (which is the only spanning tree of itself), starting from any root node with an arbitrary constant.

The next class of elementary graphs on which gradient correction is the easiest are the graphs containing a single cycle. If the graph contains n vertices, they are usually denoted by C_n (see Figure 8).

A first interesting fact easy to compute by hand and to understand mathematically, although not the central subject, is to use the orthogonal projection formula onto the space generated by cycles which, in this case, has dimension 1. It has well-known analytical expression $X(X^TX)^{-1}X^T$ where

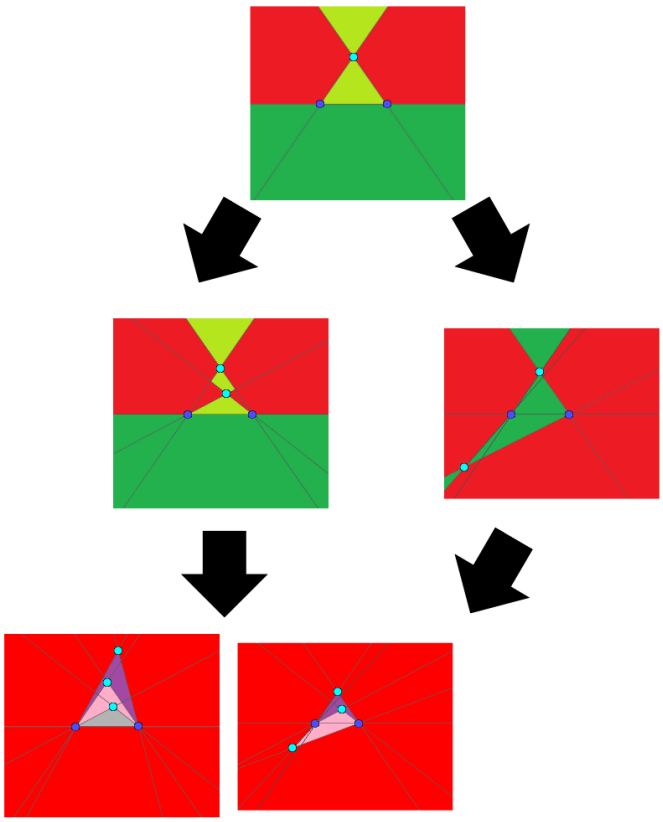


Figure 6: The topmost figure shows the first three vertices mentioned in the proof, and in green shades are the regions where the fourth vertex belonging to s_2 may be drawn. The second row shows two typical examples of a drawing with four vertices (one for each shade), where shades of green indicate the possible positioning of the fifth vertex in s_2, leading in all cases to the impossibility of drawing the remaining sixth vertex in s_1 without inducing a crossing (last row, with two typical examples of graphs with five vertices and in which drawing the last one in s_1 necessarily induces a crossing). Vertices of s_1 are displayed in dark blue whereas those of s_2 are shown in light blue.

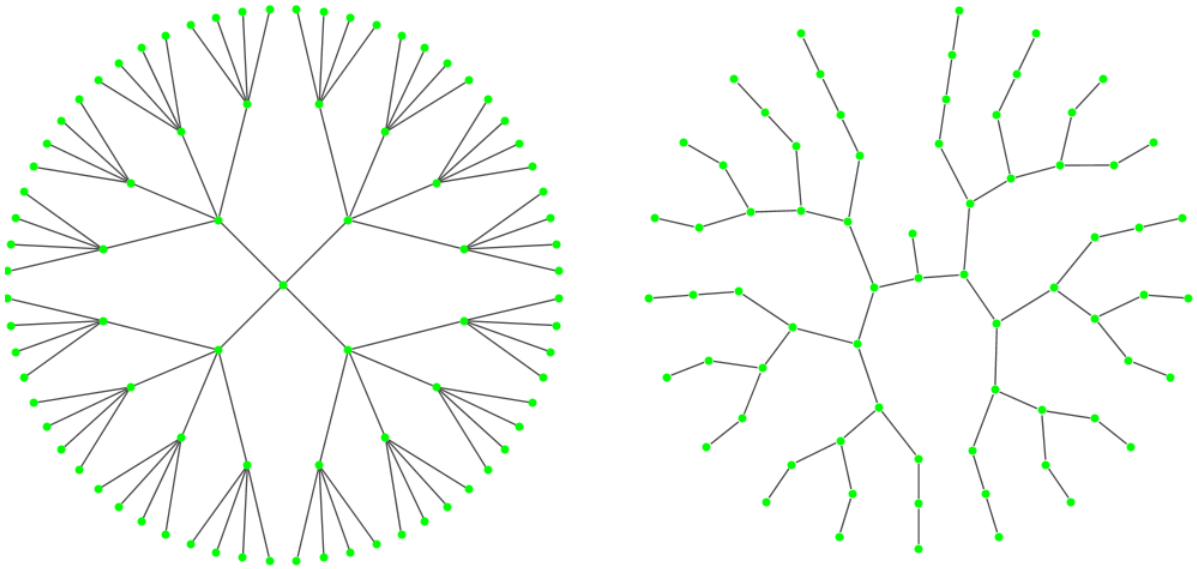


Figure 7: Examples of trees, on which gradient correction is trivial.

the rows of X^T correspond to the cycle basis. In this case, the cycle basis is simply the cycle itself, so X^T is just a transposed vector corresponding to it. In this particular case, assuming our cycle is C_n, we have $X^T X = ||X||^2 = n$ which is a scalar and $(X^T X)^{-1} = \frac{1}{n}$. Assuming our estimated gradient is given by $\vec{f}^T = (f_1, \ldots, f_n)$, then the corrected gradient $\vec{f}_{\text{cor}}$ can be obtained by subtracting to it its orthogonal projection onto the cycle space:

$$\vec{f}_{\text{cor}} := \vec{f} - X(X^T X)^{-1} X^T \vec{f}$$

Using our previous analytical expression and derivation and assuming the edges are directed clockwise or counterclockwise, one notes that this is equivalent to subtracting the mean of the components of $\vec{f}$ componentwise. It is then trivial to check that the obtained gradient is indeed irrotational as a sanity check. We also note that we derived from linear algebra the result usually obtained through elementary calculus that the mean $\overline{f}$ of $f_1, \ldots, f_n$ is the number minimizing $\sum_{i=1}^{n}(f_i - \overline{f})^2 = ||\vec{f}_{\text{cor}}||^2$, because orthogonal projections minimize the L_2 distance between the input and the projected output. In other words, we corrected the estimated gradient by finding by hand the only solution minimizing the L_2 norm. In particular, one can note that the solution is unique, and it remains so for any graph using the analytical formula for $\vec{f}_{\text{cor}}$.

This is interesting because the uniqueness of the solutions is not preserved when minimizing the L_1 norm instead of the L_2 norm, and contrary to the L_2 case, there is no analytical formula. However, in the case of a single cycle, the problem is easily solved by hand. Suppose we have the same vector $\vec{f}$ as a gradient estimate and it yields residue (say, without loss of generality) $\text{Res} > 0$. Then it is trivial to check that the solutions $\delta^{\pm}$ minimizing the L_1 norm is given by $\delta^- = 0$ and $\delta^+ \geq 0$ such that $\sum_{\text{edges}} \delta^+ = \text{Res}$. In particular, the solution is clearly not unique anymore.

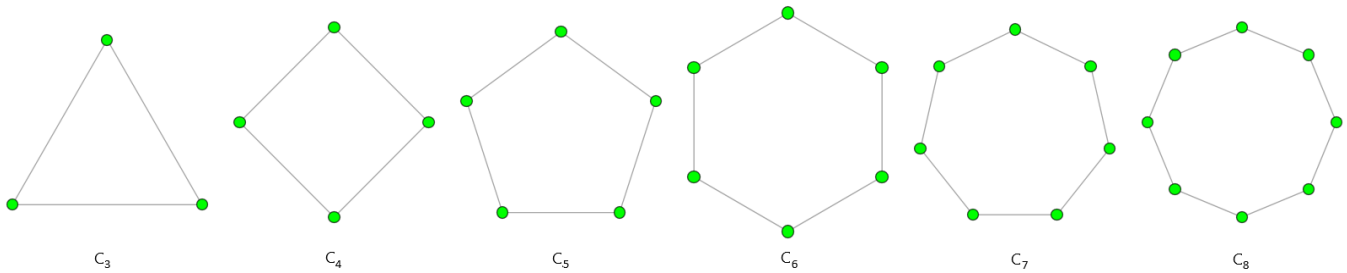
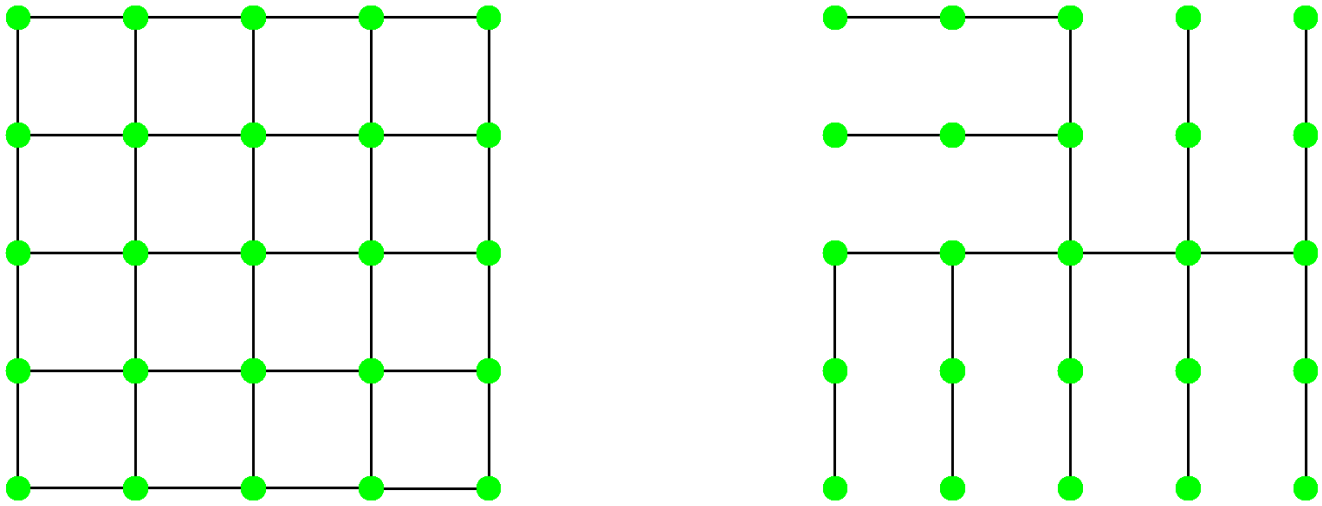


Figure 8: Examples of cycles C_n on which gradient correction is easy to understand for $3 \leq n \leq 8$. The cases $n = 1$ and $n = 2$ are not treated because self-loops C_1 make no sense in our framework, and cycles C_2 are not involved because we only consider antisymmetric gradient estimates.

3.5.3 Construction of a fundamental cycle basis



(a) Graph of a regular 5×5 grid.

(b) Example of a spanning tree of graph (a)

Figure 9: Grid graph (left) along with one of its spanning trees (right).

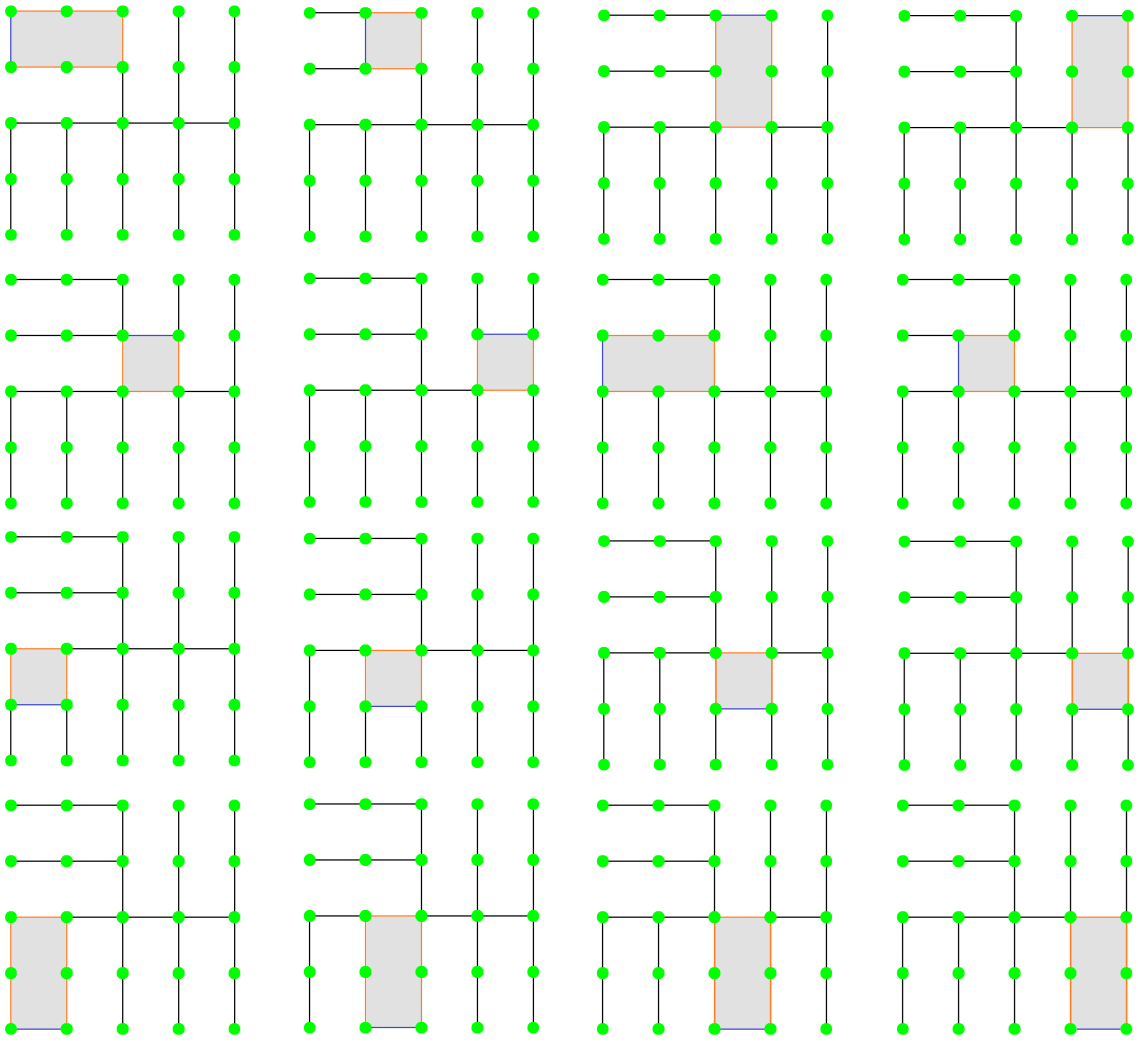
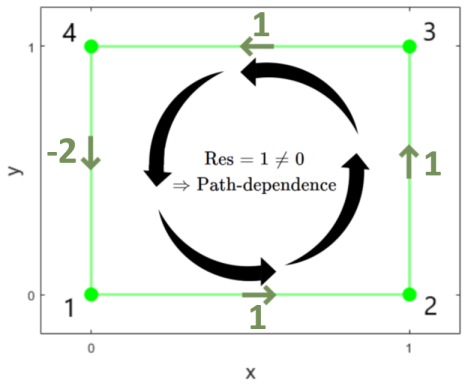


Figure 10: The 16 cycles forming the fundamental cycle basis associated to the spanning tree of Figure 9

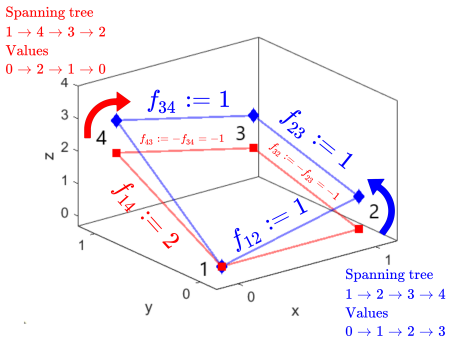
This subsection is devoted to a graphical example of the construction of a fundamental cycle basis, starting from an arbitrary spanning tree on the 5×5 grid, which is a very intuitive graph (see Figures 9 and 10). Note that the 16 elementary squares obviously form a cycle basis, which are the equivalent of the Delaunay simplices in the case of a (less intuitive) Delaunay triangulation, so as a reality check, the basis sought has cardinality 16, which also matches the usual formula $m-n+1 = 40-25+1 = 16$. Using a Delaunay triangulation instead of the grid would yield similar results.

3.5.4 An illustration of inconsistent estimates leading to the loss of path-independence

In this section, a simple example of how inconsistent estimates lead to the loss of the path-independence property of gradients is given. Consider the graph of a square whose vertices are numbered from 1 to 4 counterclockwise. Assume $f_{12} = f_{23} = f_{34} = 1$ and $f_{14} = 2$. This leads to an inconsistency, because integrating following the spanning tree $1 \to 2 \to 3 \to 4$ starting from constant 0 yields values $0, 1, 2, 3$ for the vertices $1, 2, 3, 4$, respectively. However, choosing the spanning tree $1 \to 4 \to 3 \to 2$ yields values $0, 0, 1, 2$ instead. A visualization is available in Figure 11, where the z-axis represents the estimated original function starting from constant 0 at vertex 1. The blue graph corresponds to integration on the first spanning tree and the red graph corresponds to integration on the second spanning tree.



(a) Graph of the underlying square along with its nonzero residue, leading to the loss of path-independence upon integration.



(b) Examples of inconsistent integrations starting from root vertex 1 and value 0. No global constant can be added or subtracted to make the two graphs overlap completely.

Figure 11: Square graph (left) along with two inconsistent integrations due to path-dependence (right).

4 Experiments

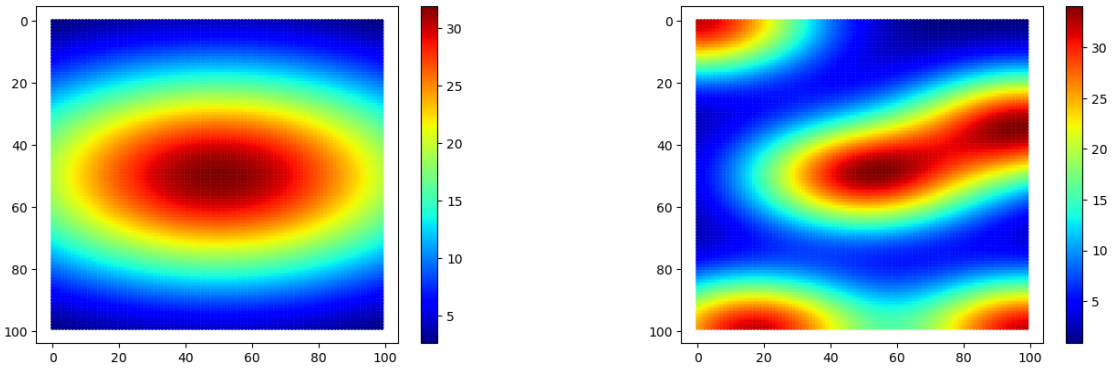
We conducted experiments both on artificially generated data and real data. The first experiments were carried out on toy models such as a two-dimensional Gaussian to model e.g. a hill or several Gaussians to simulate a mountainous area with passes, valleys and peaks. The advantage is that the experiments are easy to generate and easier to understand due to the relative smoothness of the simulated terrain. The second set of experiments tested the algorithms on a few arbitrary images such as paintings. The point was to examine the behavior of the algorithms in complex cases where many discontinuities arise, even if the data is not related to an elevation profile. The third set of experiments was carried out on realistic InSAR topographic phase simulation data containing mountainous terrain such as Mount Zeil, Mount Sinai, etc. Details of the simulation can be found in [6]. The fourth set of experiments involved simulated terrains with various levels and types of noise (Gaussian, impulsive, plane cuts, atmospheric etc.). Then, edge cases are discussed, followed by a fifth set of experiments conducted on interferograms and where temporal consistency was tested.

The last set of experiments concerns generic gradient correcting, tested essentially on a grayscale image of the Mona Lisa with various noise levels added to its original gradient.

In order to have enough comparable data sets, one can note that modulating a phase profile mod 2π is equivalent to modulating an elevation profile mod $\frac{\lambda}{2}$. Since the behavior of the algorithms depends mainly on the magnitude of the gradients, it can be interesting to compare them on the same terrain up to a contraction or a dilation. This is equivalent to simulating an elevation terrain once and using a parameter $h > 0$ as a characteristic length instead of π to modulate mod $2h$. By changing h, one can monitor the L_1 error (defined formally as the difference between the unwrapped data and the ground truth, minus the median of this difference, as we are working up to a constant, taken in absolute value) as a function of h or of the ratio of wrong gradient estimations, i.e. as the relative steepness of the terrain varies.

$$\text{L}_1 \text{ Error } := ||\text{Ground truth} - \text{Unwrapped signal} - \text{Median}\,(\text{Ground truth} - \text{Unwrapped signal})||_1.$$

4.1 Toy models



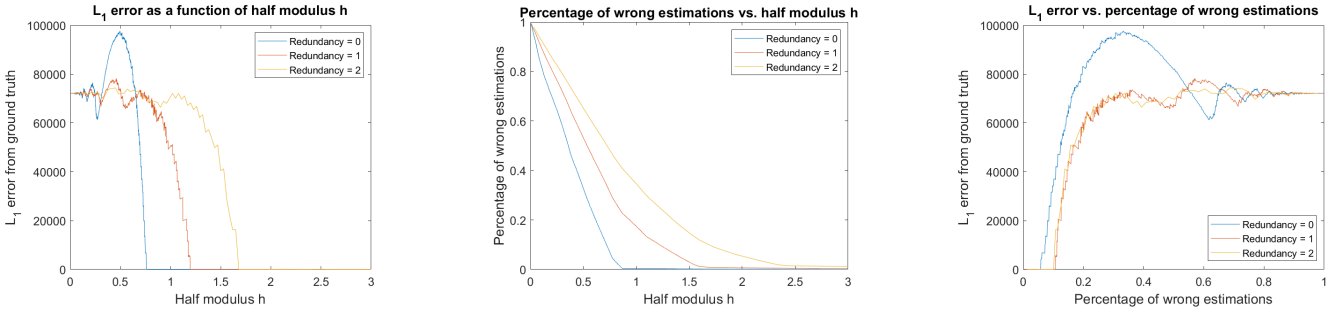
(a) Most basic toy model: a simulated hill.

(b) Toy model: a simulated mountainous area.

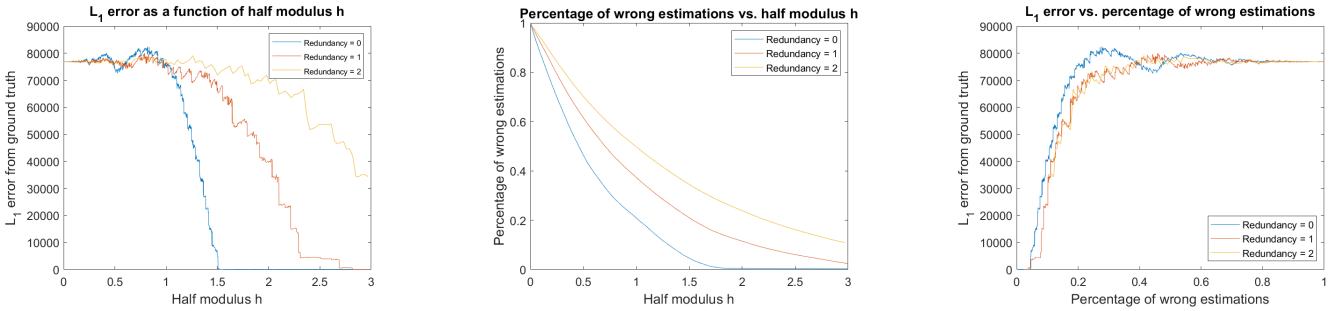
Figure 12: Simulated toy models.

Figure 12a is the first model on which the unwrapping algorithms were tested. The range of values goes roughly from 0 to 30 and the characteristic lengths h which were used lie typically between 0 and 3. The larger the characteristic length, the smaller the wrong gradient estimation ratio and the better the unwrapping. The results can be seen in Figures 13a for redundancies $r \in \{0, 1, 2\}$.
In particular, Figure 13a shows that while there is indeed an information gain at a given wrong estimation ratio of the gradients going from $r = 0$ to $r > 0$ (but not from $r = 1$ to $r = 2$) as shown by the lower L_1 error at fixed ratio (rightmost figure), the figure in the middle shows that at a given characteristic length h, more gradient estimation errors are made with higher redundancy. The figure on the left shows that the information gain shown on the rightmost figure is not enough to offset this increase in errors (at given h, more L_1 error is made with higher redundancy, except when h is so small that all the unwrappings have failed and comparison is not relevant).

Similar observations can be made when simulating a more elaborate terrain (see Figure 12b and Figure 13b), although the information gain using redundancies $r > 0$ is smaller. One notices in all cases a phase transition (i.e. sudden change of behavior) of the error depending on parameter h or the wrong estimation ratio. This causes the curve to suddenly increase in a step-like fashion with breakpoints. A visual representation of an example of a phase transition can be seen in Figure 14, precisely by comparing the first and the third picture from the second row around the two peaks in the center.



(a) Experiment result for terrain of Figure 12a.



(b) Experiment result for terrain of Figure 12b.

Figure 13: From left to right: L_1 error vs. half-modulus, percentage of wrong gradient estimations vs. half-modulus, L_1 error vs. percentage of wrong gradient estimations; associated to the elevation profiles in Figure 12

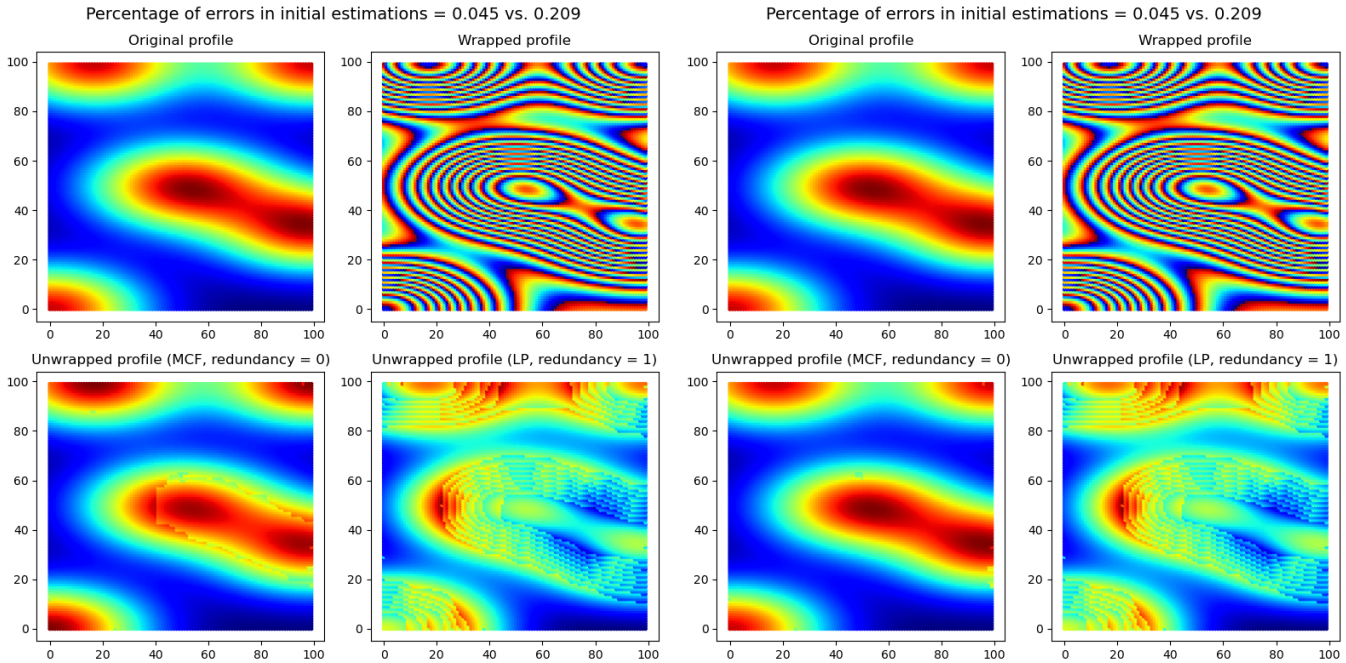


Figure 14: The four leftmost figures: h = 1.505, just before a transition; the four rightmost figures: h = 1.506, past a transition, when r = 0. Note how a whole region around the two peaks in the middle of the image was corrected using MCF (left, bottom left vs. right, bottom left). Meanwhile, the unwrapped signal when r = 1 is still far from the ground truth (left, bottom right vs. right, bottom right). The percentages of errors in initial estimations correspond to the proportions of wrongly estimated gradients for r = 0 and r = 1, respectively. They are the same in the two blocks because the change in h is so slight that the percentages have not been affected up to a precision of 10^{-3}, yet this change was sufficient to entail a clearly noticeable correction when r = 0.

In the unwrapped figure with no redundancy, a "step" can be visualized where a small erroneously unwrapped region (left, $h = 1.505$) ends up correctly unwrapped (right, $h = 1.506$). The change in h is so slight that the percentage of errors has not changed meanwhile in an amount greater than 10^{-3}. This shows the abruptness of the unwrapping process. Besides, the figure also shows for the same parameter h (but not the same errors in wrong estimations) the evolution of the unwrapping process with redundancy 1, which is far behind the one with redundancy 0. In other words, unwrapping with no redundancy is far more efficient in this case, and actually in most cases, as more examples will confirm.

4.2 Paintings

In this subsection, a single painting was studied: that of Mona Lisa, see Figure 15. Using the same methodology as that of the previous subsection, one obtains Figure 16. This time, no information gain occurs (see the rightmost figure) as the error curve follows the same trend depending on the wrong estimates ratio independently of r. However, more errors are made with higher redundancy and this results in a worse unwrapping (see the leftmost figure). It seems like the more intricate the terrain or figure to be unwrapped, the less information is gained through nonzero redundancy.

Figure 15: Painting of Mona Lisa

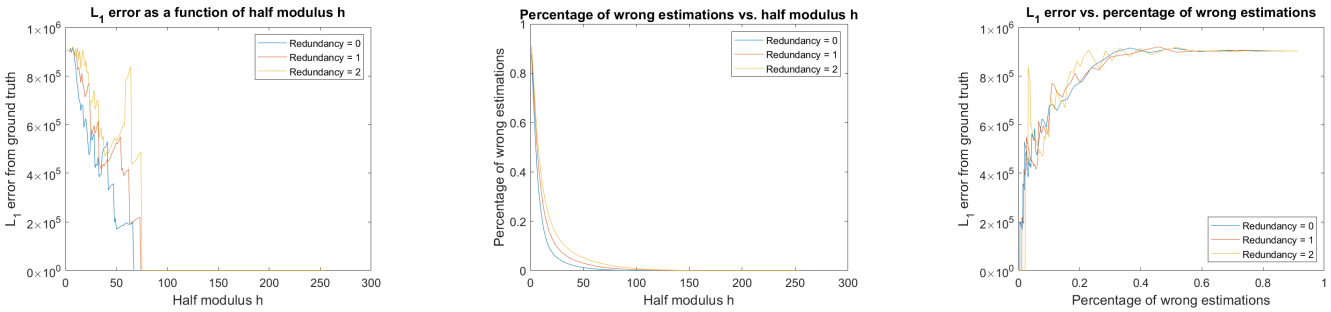
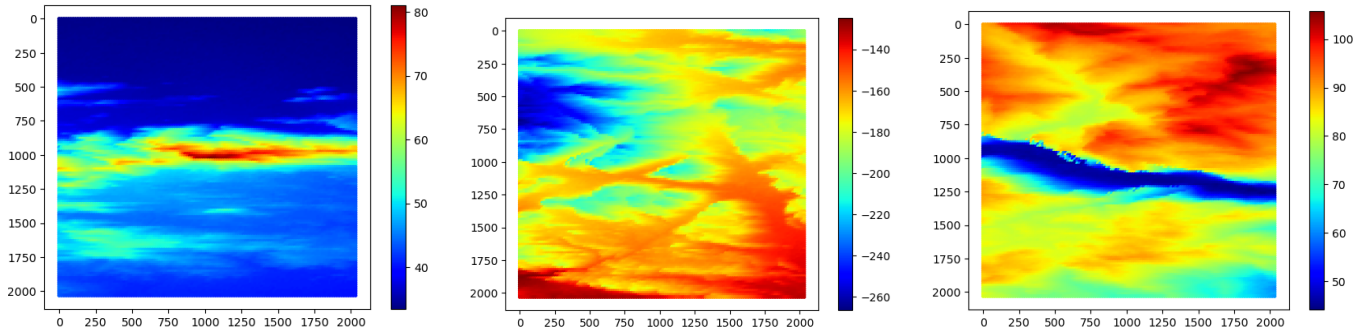


Figure 16: From left to right: L_1 error vs. half-modulus, percentage of wrong gradient estimations vs. half-modulus, L_1 error vs. percentage of wrong gradient estimations; associated to the elevation profile in Figure 15.

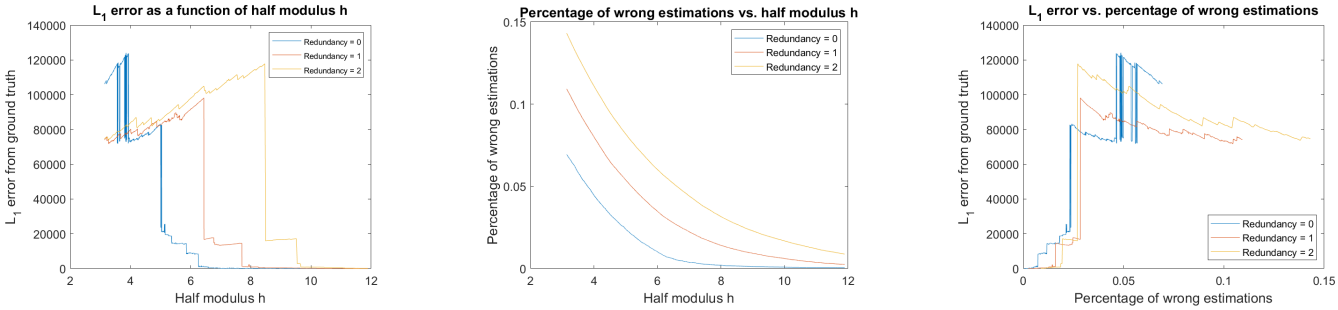
4.3 Realistic terrain simulations

In this subsection, attempts to unwrap realistic simulated terrain profiles such as Mount Zeil, Mount Sinai and El Capitan, using the same methodology, have been made. The ground truth elevation profiles can be seen in Figures 17a, 17b and 17c.

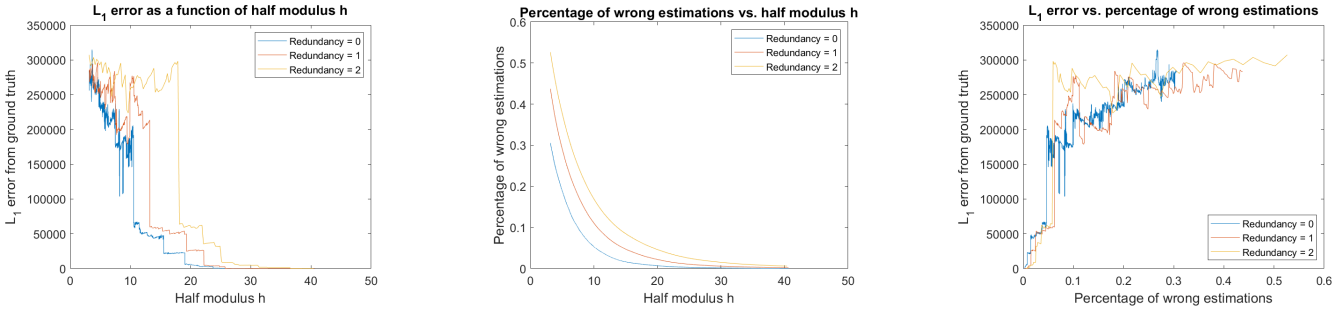


(a) Elevation profile of Mount Zeil (b) Elevation profile of Mount Sinai (c) Elevation profile of El Capitan

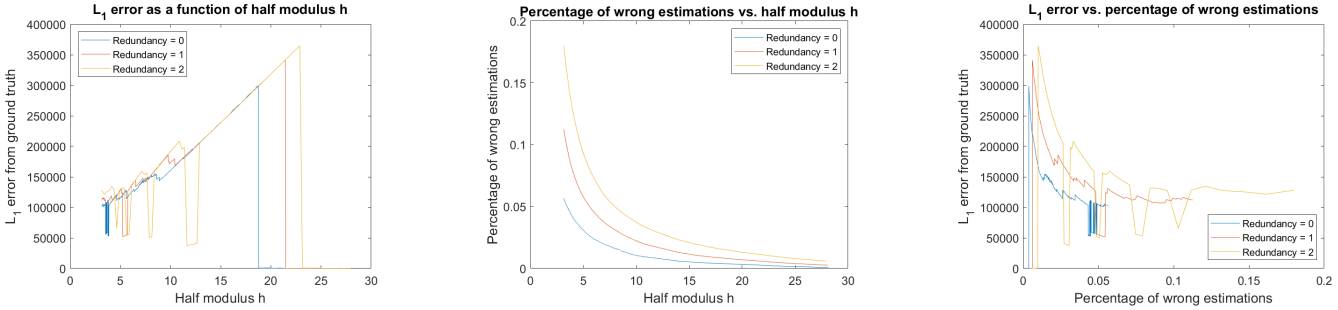
Figure 17: Realistic terrain simulations.



(a) Experiment result for terrain of Figure 17a.



(b) Experiment result for terrain of Figure 17b.



(c) Experiment result for terrain of Figure 17c.

Figure 18: From left to right: L_1 error vs. half-modulus, percentage of wrong gradient estimations vs. half-modulus, L_1 error vs. percentage of wrong gradient estimations; associated to the elevation profiles in Figure 17

Each profile has its own characteristics, but contrary to basic toy models, the terrains are relatively

rough and discontinuous. However (see Figure 18), in all cases, the same scenario occurs: when errors are small, very slight gains in information are made (on each plot triplet, when the wrong estimations ratio is small enough and the error reasonable [a zoom-in might be necessary as it is very close to zero], the rightmost plot exhibits error curves such that the curve with no redundancy is above the ones with redundancy) but significantly more errors are also made at fixed parameter h (this is immediate from the plot on the middle), leading to a worse unwrapping result overall (see the leftmost plot, the error curve vs. parameter h explodes at larger values with redundancy than without, meaning that no redundancy offers more flexibility and indicates that it is generally a better choice).

4.4 Noisy toy models

In this subsection, the effect of various types of noise (Gaussian, impulsive, plane cuts, atmospheric) and intensities have been studied.

The first case is the one involving centered Gaussian noise. Every point of the toy model in Figure 12b had its value bumped up or down by a Gaussian random variable with a given standard deviation σ. We varied h, and for each h a new noise realization was made, leading to a "thick" or rapidly oscillating curve instead of a smooth one due to noise variations. This process is more useful than choosing a specific noise realization and varying h as this would show that redundancy is deleterious in one noise realization, whereas the experiment in Figure 19 shows that the error is always larger with redundancy than without in all noise realizations.

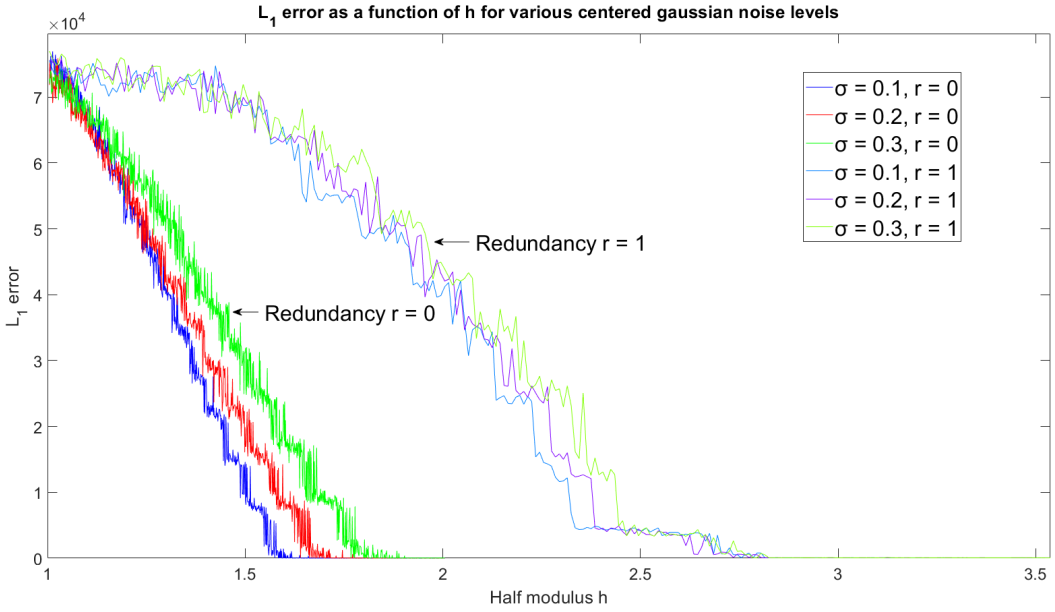


Figure 19: L_1 errors vs. parameter h for noisy toy model realizations (Gaussian noise)

In particular, the graph 19 shows that it is harder to unwrap a noisier (and thus rougher) terrain (the higher σ, the higher the error magnitudes at a given redundancy level).

The second case involves impulse noise. We shall proceed in the same way as the first case, i.e. a new noise realization for each new h on toy model 12b. There are two subcases: the first one involves a percentage of points selected uniformly at random which received an impulse noise of $\pm$ a fixed amplitude, where the amplitude is a nonnegative real number. An example of the experiment is

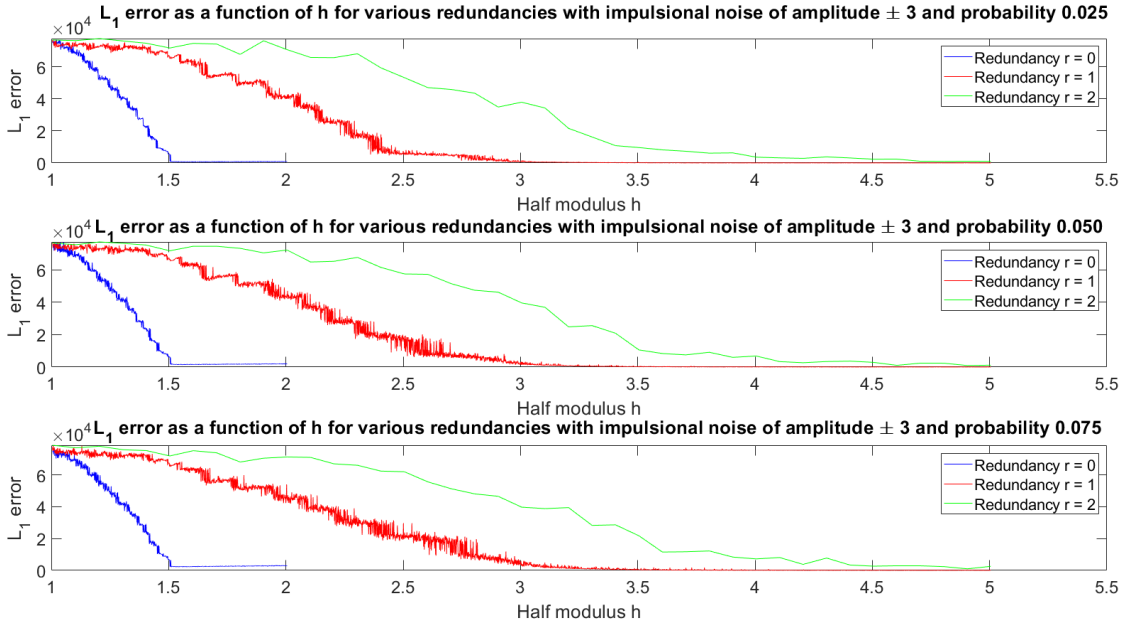


Figure 20: L_1 errors vs. parameter h for noisy toy model realizations (Impulse noise)

given in Figure 20 with amplitude 3 and several percentages. The second subcase involves impulse noise which cuts the plane with a line of width 1 (i.e. impulse noise with the same sign was added to a line of points in a rectangular array, cutting it into several disconnected components). An example of this experiment is given in 21 with amplitude 3 and one or several cuts.

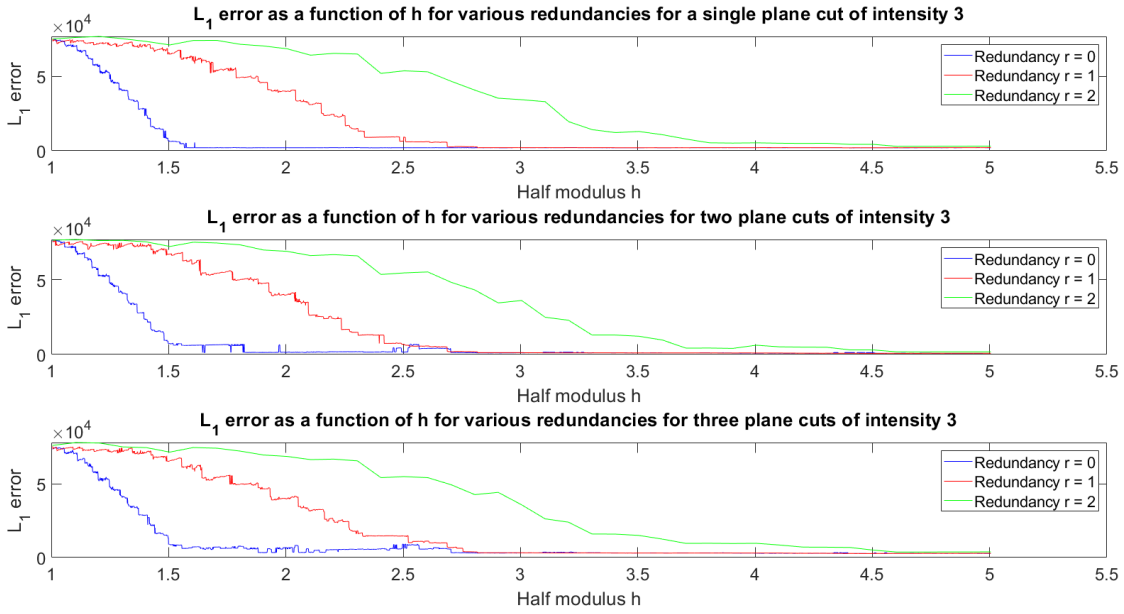


Figure 21: L_1 errors vs. parameter h for noisy toy model realizations (Plane cut noise)

In both cases, the error grows again almost systematically with the redundancy. Essentially, the same observation is made as the one in the first case.

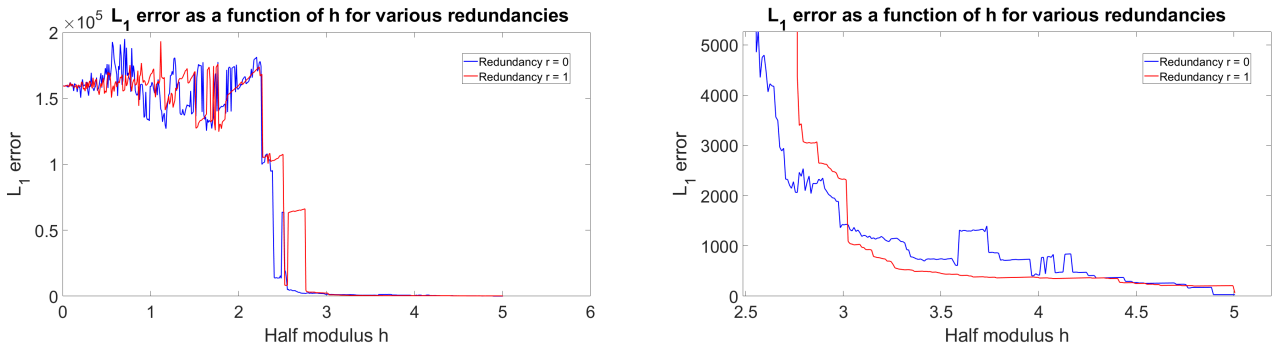


Figure 22: L_1 errors vs. parameter h for simulated atmospheric noise (left: full graph, right: zoom-in)

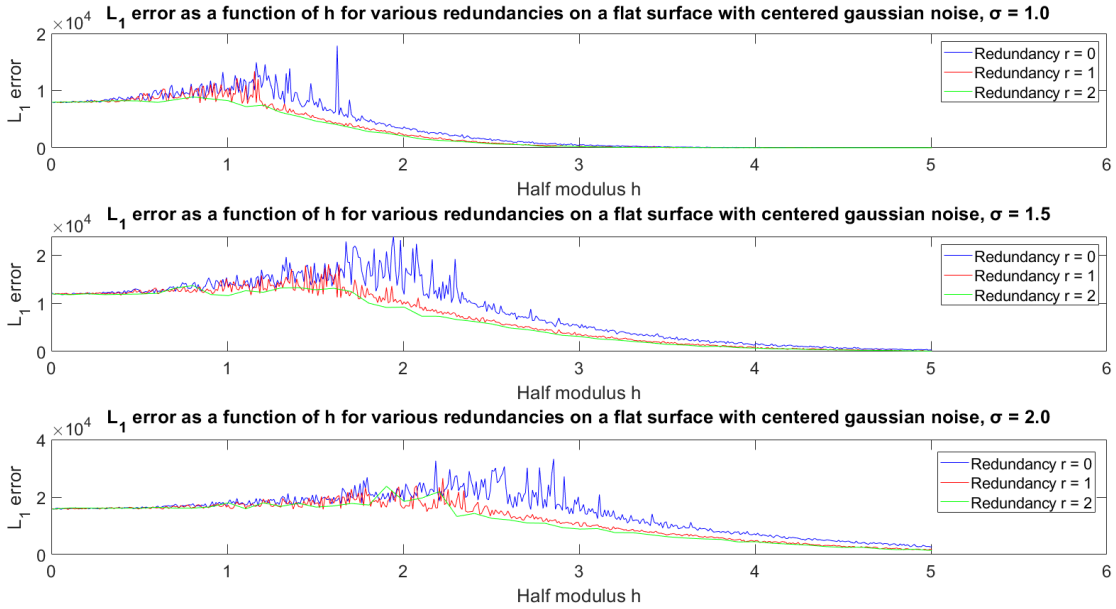


Figure 23: L_1 errors vs. parameter h for noisy flat model realizations (Gaussian)

The last case involves atmospheric noise. In this simulation, a stack of 29 Sentinel-1 images acquired over Teramo in 2023, for orbit 44, covering about 874 km^2, was used to determine the positions of the PSs. A point is considered a PS when its amplitude dispersion index (relative amplitude variation in time) is relatively small. The amplitude dispersion index is defined as

$$D_a := \frac{\sigma_a}{\mu_a},$$

where μ_a and σ_a are the mean and standard deviation of the amplitude over time (in this case, the 29 dates). We used $D_a < 0.25$ [9] as a criterion to detect persistent scatterers (PSs). For the phase simulation, a ramp of slope $\frac{\pi}{5000}$ radians per pixel was used. We also added a fractal atmospheric noise with a fractal surface of dimension 2.67, stretched to the interval $[-8.74, 8.74]$. The code to generate this atmospheric simulation was adapted from the DORIS software [13]. The interval for the atmospheric noise complies with the literature [14], which documents a maximal variation of 2 radians per 100 km^2. Gaussian phase noise was then added to each PS, with a standard deviation $1.5D_a$. The same simulation was used for each h and the result is given in Figure 22. In particular, a zoom-in when errors are small shows that redundancy helps reduce the error

A special case deserves to be mentioned: the case where Gaussian noise is added to a flat region. In this special subcase of case 1 where toy model 12b is replaced with a flat profile, the experiment

23 shows improvement in accuracy when the redundancy is higher. In this experiment, a new noise realization is made for each h.

4.5 Edge cases

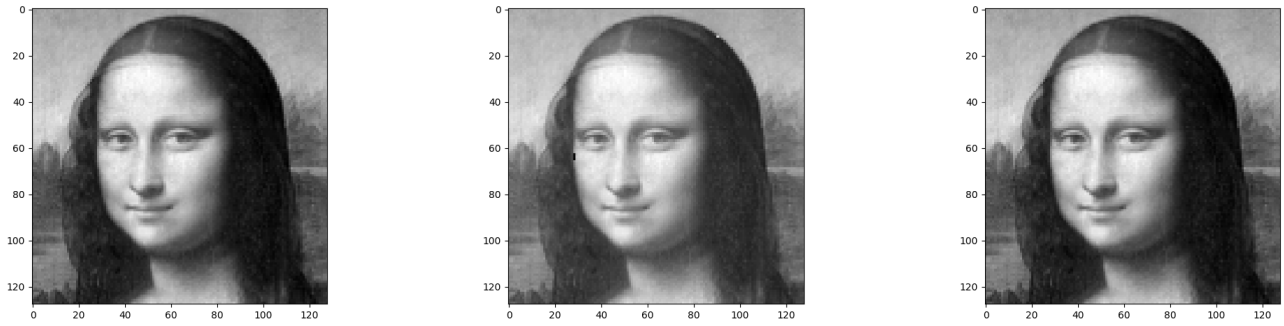


Figure 24: From left to right: original image, unwrapped image with no redundancy, unwrapped image with redundancy 1.

Even if in the vast majority of cases, redundancy worsens the unwrapping process, edge cases do exist where redundancy yields slightly better results with fine-tuned parameters. This is usually the case when the unwrapping process without redundancy is close to being perfect. Slight variations might cause the error curve with $r > 0$ to go under the one with $r = 0$ when the error is small. An instance is given in Figure 24. However, these fringe events are unpredictable without a ground truth and thus do not imply that using redundancy helps when the error is relatively small. It is actually quite rare and as a consequence, it is recommended to use no redundancy when no ground truth is available, unless one is dealing with the last cases of the previous subsection.
In the edge case of Figure 24, the leftmost picture is the ground truth, the middle image is the unwrapped image with no redundancy and the rightmost image was reconstructed with redundancy 1, both using the same method (LP). However, this event is essentially due to luck and simply shows that edge cases may exist, but not that they are usual (they are not, and by a large margin). In this case, the original image was treated as a 128×128 array of integers lying in $[0, 255]$.

4.6 Example of interferogram unwrapping

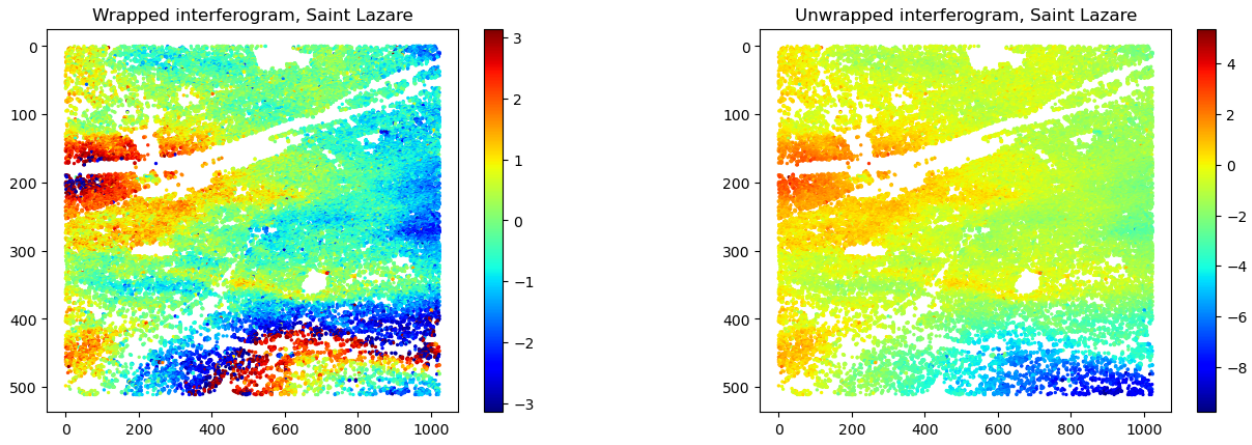


Figure 25: Sentinel-1 interferogram of Saint Lazare, Paris, obtained from dates 2018-04-15 and 2018-04-21. From left to right: wrapped profile, unwrapped profile using MCF.

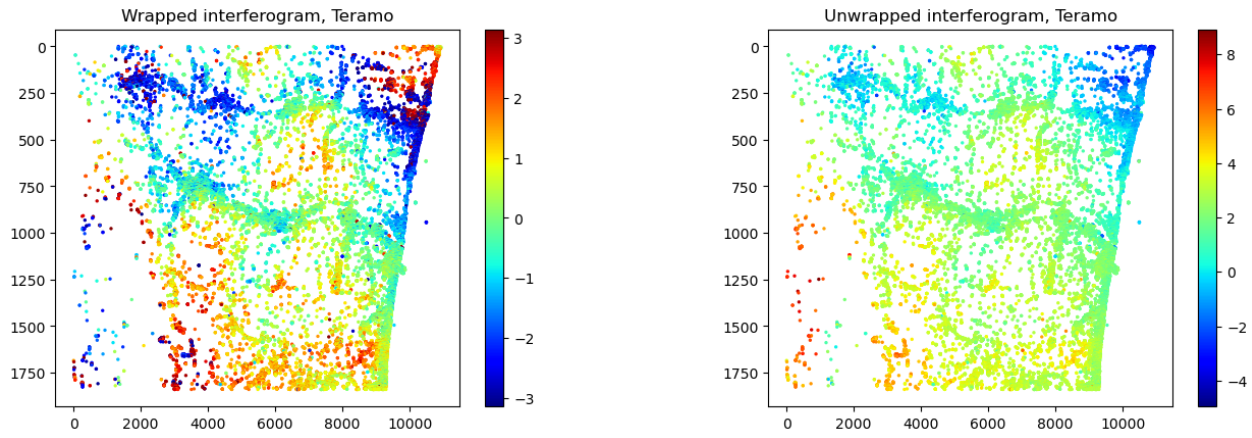


Figure 26: Interferogram of Teramo obtained from dates 2023-01-06 and 2023-01-18. From left to right: wrapped profile, unwrapped profile using MCF.

An example of the unwrapping of an interferogram where each point is a PS (i.e. a point whose amplitude dispersion is small enough, $D_a < 0.25$) is shown in Figure 25. An interferogram is a measure of terrain deformation between two dates.

Experiments on Sentinel-1 interferograms of Teramo, Abruzzo were also conducted, using 29 images of consecutive dates, labelled chronologically from 0 to 28 in an attempt to reproduce the unwrapping inconsistency experiment of article [5]. An example is shown in Figure 26.

From these images, 27 consecutive image triplets were drawn, of the form $(a, b, c) = (i, i+1, i+2)$ for $0 \leq i \leq 26$. For each of these triplets, three wrapped interferograms were deduced: $\overline{\varphi_{ab}}$, $\overline{\varphi_{bc}}$ and $\overline{\varphi_{ca}}$ using couples (a, b), (b, c) and (c, a), respectively. The corresponding unwrapped profiles φ_{ab}, φ_{bc} and φ_{ca} were computed using either MCF with zero redundancy or LP with redundancy 1 and 2. Then temporal inconsistency coefficients were deduced, defined as the percentage of nonzero elements of

$$\varphi_{ab} + \varphi_{bc} + \varphi_{ca} - \operatorname{Mode}(\varphi_{ab} + \varphi_{bc} + \varphi_{ca})$$

where a mode was subtracted to ensure the consistency of the unwrapping constants, averaged over the 27 triplets. This metric can be construed as a measure of the quality of the unwrapping process (but is actually an underestimation of the error rate because, ignoring the unlikely cases where errors cancel each other out, subtracting a mode maximizes the number of zeros). Other candidates for global constants could be the median, but it has the weakness of sometimes largely overestimating the error rate. For instance, contrary to the mode, it is not necessarily a value of the array considered, and if it is not a multiple of 2π, then we would get an error rate of 100%. In practice, and because the number of PSs is very large (tens of thousands) compared to the typical number of values assumed (a few integer multiples of 2π), the median is very often equal to the mode. This was observed to be true for all the triplets at all redundancy levels except for triplet number 15 at $k = 2$.

Another way to proceed is to compute the temporal consistency of the gradients $\nabla\varphi$ corresponding to the edges of the same spanning tree of the Delaunay graph obtained from the PSs. This permits one to do away with the unknown constant and could lead to more accurate estimates. The temporal inconsistency would simply be defined as the percentage of nonzero elements of

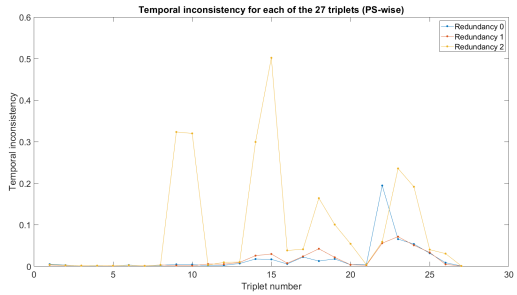
$$\nabla\varphi_{ab} + \nabla\varphi_{bc} + \nabla\varphi_{ca}$$

The results are shown in Table 1 and the detailed distributions are shown in Figure 27.

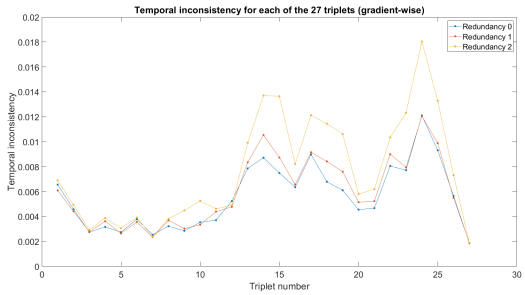
Table 1: Temporal inconsistency as a function of redundancy, PS-wise (top) and gradient-wise (bottom).

Redundancy	Mean temporal inconsistency	Median temporal inconsistency	Standard deviation
0	0.0184	0.0048	0.0386
1	0.0154	0.0048	0.0198
2	0.0906	0.0307	0.1342

Redundancy	Mean temporal inconsistency	Median temporal inconsistency	Standard deviation
0	0.0056	0.0052	0.0026
1	0.0060	0.0052	0.0028
2	0.0076	0.0062	0.0043



(a) Temporal inconsistency, φ-wise.



(b) Temporal inconsistency, $\nabla\varphi$-wise.

Figure 27: Temporal inconsistency for each of the 27 triplets, numbered chronologically, for each level of redundancy, PS-wise (left) and gradient-wise (right).

According to these metrics, and over the elevation profiles of Teramo, similar performances can be observed when using MCF with redundancy 0 as opposed to LP with redundancy 1 (and exactly the same when considering the median error using both methods), but the results are significantly worse for LP with redundancy 2. While there seems to be a non-negligible relative improvement using redundancy 1 instead of redundancy 0 in terms of mean error PS-wise, one ought to look at the distribution of Figure 27 and note that the number of outliers is small for these redundancies, so it makes sense to ignore them by prioritizing the median instead. Also, one has to keep in mind that a gradient error can induce PS-wise errors in a large region, so the PS-wise distribution is probably less accurate and more likely to be affected by statistical anomalies, leading to outliers, so the gradient-wise metric might be more relevant. However, while the degree to which these metrics are correlated with unwrapping quality could be called into question, the associated results seem to match those associated with most of the previous experiments, showing the deleterious effects of nonzero redundancy with a much more straightforward approach (directly comparing with a ground truth by computing the L_1 error).

To understand better how temporal inconsistency helps pinpoint errors when unwrapping interferogram triplets without a ground truth at hand, a visual proof follows. The errors are clearly visible in the (PS-wise) temporal inconsistency graph of Figure 30 as points with nonzero temporal inconsistency.

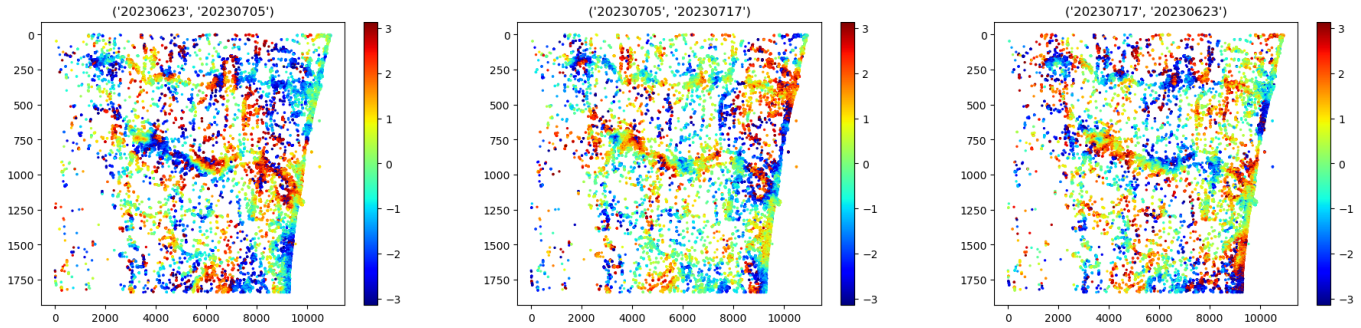


Figure 28: From left to right: wrapped interferograms between dates 2023-06-23/2023-07-05, 2023-07-05/2023-07-17 and 2023-07-17/2023-06-23.

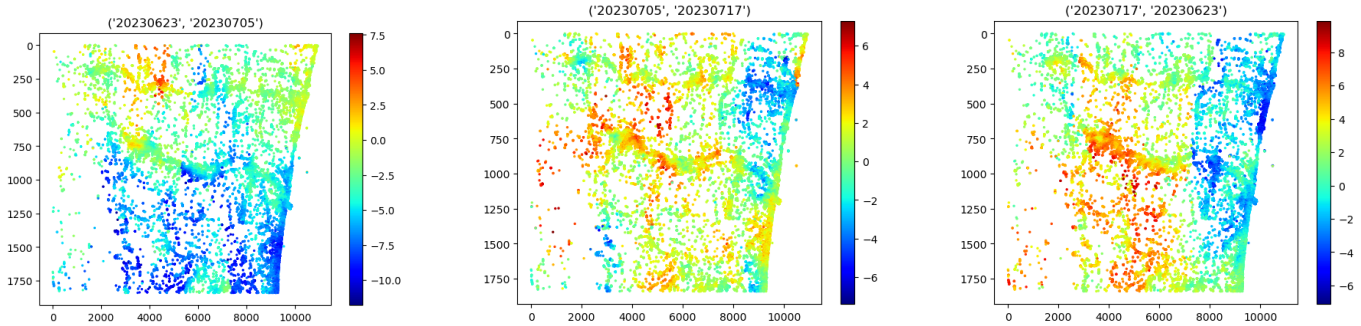


Figure 29: From left to right: unwrapped interferograms between dates 2023-06-23/2023-07-05, 2023-07-05/2023-07-17 and 2023-07-17/2023-06-23, with redundancy 2.

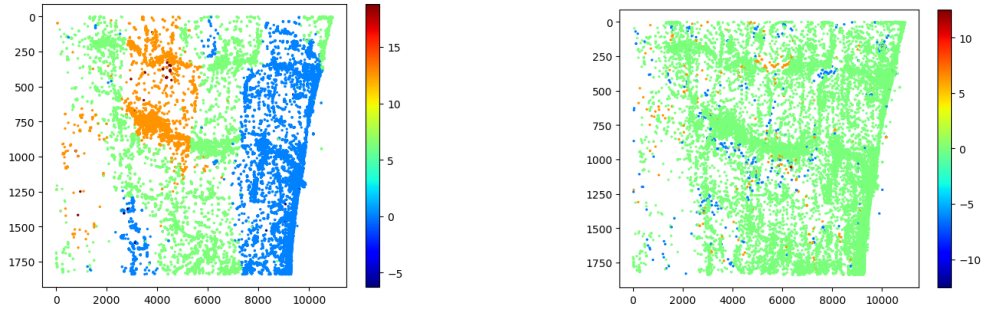


Figure 30: Left: Temporal inconsistency in the interferogram triplet with the worst result PS-wise (50%). This means the error rate is at least $\frac{1}{2}$. Note that temporal inconsistency is necessarily an integer multiple of 2π, and that while (PS-wise) temporal inconsistency is supposed to be zero, this is up to an unknown constant, so it is not possible to be sure where the errors lie. Having more than a single color simply shows there must be errors, with correctly unwrapped points belonging to a single color. Right: corresponding gradient-wise temporal inconsistency (where each point corresponds to an edge of the spanning tree and was placed in the middle of it). Since they are not up to a constant, a properly unwrapped (error-free) gradient must show up as zero on the graph, otherwise, an unwrapping mistake has been made. Note that a few gradient-wise errors can induce PS-wise errors in a large region.

The example of Figure 30 is a special case where significant errors were made (by far the worst result of the unwrapping process for the 27 triplets with redundancy $0, 1$ and 2, with a temporal inconsistency of approximately $\frac{1}{2}$ PS-wise). It was deliberately cherry-picked to show the usefulness of temporal inconsistency.

Figures 31 and 32 showcase a comparison of the worst (PS-wise) temporal inconsistency results

for each redundancy level $0, 1, 2$ across the 27 triplets.

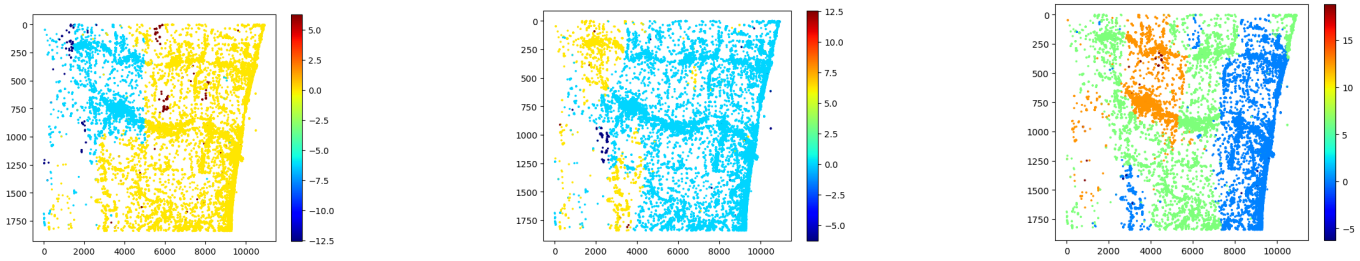


Figure 31: From left to right: worst temporal inconsistency results PS-wise (19%, 7%, 50%) across the 27 triplets for redundancy levels 0, 1 and 2, respectively.

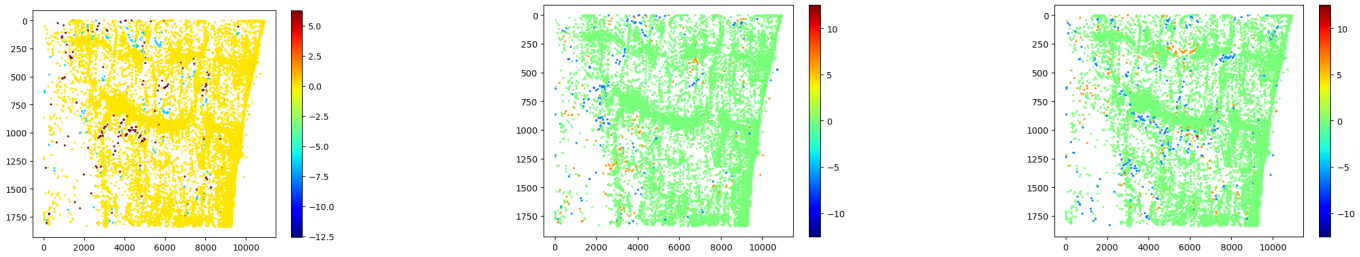


Figure 32: From left to right: gradient-wise inconsistency corresponding to the scatter plots of Figure 31.

4.7 Generic gradient corrector

As was discussed briefly in subsection 3.4, it might happen that one wishes to correct an approximate gradient on a graph which is a gradient plus some global, moderate centered Gaussian noise and some local, sparse but severe impulse noise. To do so, it was explained that the aforementioned LP method simply works without congruency mod 2π, and MCF can also be used as long as one exercises caution due to the way OR-Tools handles flows which must be integers. The residues (integration of the noisy gradient around each cycle of a cycle basis) just aren't integer multiples of 2π anymore, and so the δ also aren't so anymore. However, the result is still the closest gradient in L_1 norm. The impulse noise justifies the use of the L_1 norm which is less sensitive to outliers. If impulse noise were not present, the L_2 metric would suffice and the theory would be much more elementary, as the closest gradient in L_2 norm is simply its orthogonal projection onto the space of gradients, which is the image of matrix X^T if X is the node-arc incidence matrix of the graph. The projector has the well-known analytical formula $X^T(XX^T)^{-1}X$, though its computation might prove computationally costly.

The notable results can be found in Figure 33 for both MCF and LP, with various amounts of redundancy and noise levels.

If one has a priori knowledge of the noise levels, using confidence weights (or costs) proportional to the inverse square of the noise intensity aids considerably the denoising process. The square exponent is used to exclude outliers bound to provide wrong information. It can be shown empirically that an exponent of 2 works best in most situations. A table equivalent to Figure 33 using these weights is shown in Figure 34.

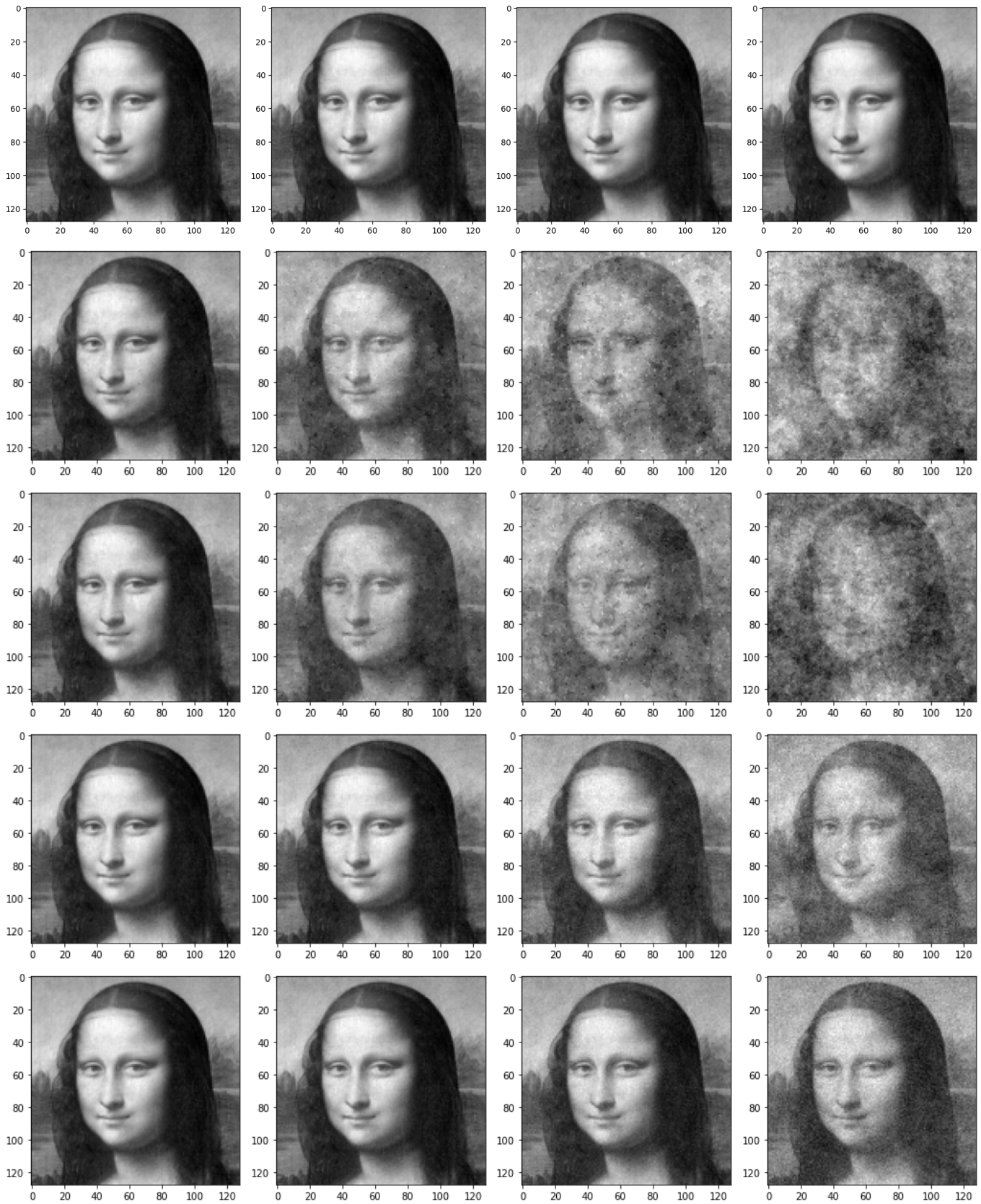


Figure 33: Results for generic field reconstitution. Rows from top to bottom: original picture, reconstitution using MCF (redundancy $r = 0$ and $\epsilon = 10^{-3}$), reconstitution using LP with redundancies $r = 0$, $r = 1$ and $r = 2$, respectively. Columns from left to right: low noise level (impulses of intensity ± 25 with probability 0.1 and centered Gaussian noise with $\sigma = 5$) moderate noise level (impulses of intensity ± 50 with probability 0.2 and centered Gaussian noise with $\sigma = 10$), high noise level (impulses of intensity ± 100 with probability 0.25 and centered Gaussian noise with $\sigma = 20$) and very high noise level (impulses of intensity ± 100 with probability 0.25 and centered Gaussian noise with $\sigma = 50$). For reference, the Mona Lisa picture is a 128×128 array of integers ranging from 0 to 229.

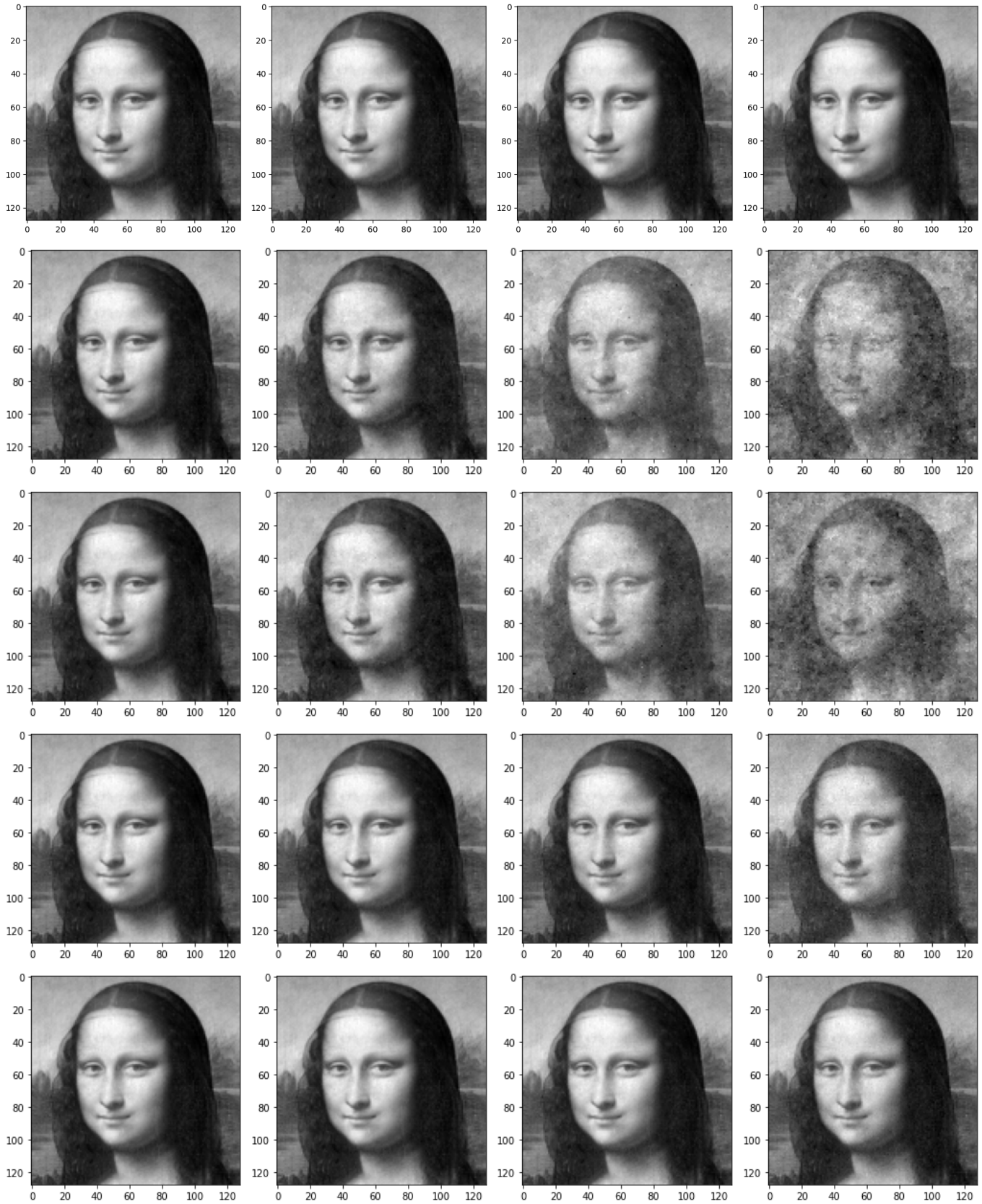


Figure 34: Results for generic field reconstitution, using a priori knowledge of noise levels. Rows from top to bottom: original picture, reconstitution using MCF (redundancy $r = 0$ and $\epsilon = 10^{-3}$), reconstitution using LP with redundancies $r = 0$, $r = 1$ and $r = 2$, respectively. Columns from left to right: low noise level (impulses of intensity ± 25 with probability 0.1 and centered Gaussian noise with $\sigma = 5$) moderate noise level (impulses of intensity ± 50 with probability 0.2 and centered Gaussian noise with $\sigma = 10$), high noise level (impulses of intensity ± 100 with probability 0.25 and centered Gaussian noise with $\sigma = 20$) and very high noise level (impulses of intensity ± 100 with probability 0.25 and centered Gaussian noise with $\sigma = 50$). For reference, the Mona Lisa picture is a 128×128 array of integers ranging from 0 to 229.

4.8 Comments

4.8.1 On phase unwrapping (with congruency)

It was shown that the algorithms tend to respond abruptly to the wrong gradient estimation ratio at arbitrary values depending on the terrain under consideration and the redundancy. The error curves exhibited all include regions with small variations where the error magnitude is almost constant, and sudden large, step-like variations. In toy models, the error curve tends to look like a curve exhibiting a phase transition, namely a sudden change of behavior, at a given threshold whose value depends on the terrain under scrutiny, below which order dominates (L_1 error close to zero) and above which large errors occur. In the former region, the curve is almost constant and equal to small variations above but close to zero, whereas in the latter, the curve looks like a power-law when smoothed out, and zooming in, small steps can be observed, corresponding to mini-transitions and giving it the appearance of a tiny staircase.

What is actually happening is an information-theoretic phase transition such that below a critical threshold in wrong gradient estimates the profile is reconstructed without errors and above which large regions of the plane delimited by discontinuities are erroneously bumped up (or down) by a correction of $2h$, leading to a curve looking like a staircase when zoomed in, and a power law when zoomed out, past the critical threshold.

In paintings and real data, where relative order and continuity might be lost, the error curve loses its well-behaved, almost analytical form but step-like discontinuities still occur in all cases. In particular, there is still a critical threshold below which the reconstruction is almost perfect and above which large errors occur. In most cases, the critical threshold lies between 0% and 5% of wrongly estimated gradients. Ignoring edge cases where the gradients are equal to h, this ratio corresponds to the percentage of gradients larger than h in absolute value. Usually, the more intricate the image (presence of discontinuities or large variations/dynamic range of values, etc.), the smaller the critical threshold. Since the ratio of wrong estimates tends to depend relatively smoothly on h, a similar behavior and critical threshold can also be seen when plotting the error as a function of h (though order dominates past this new threshold instead of before, since the ratio of wrong estimates is a decreasing function of h).

In particular, unwrapping results were compared depending on the redundancy parameter r. It was shown that while redundancy helps recover information at a given wrong estimation rate (lower L_1 error), the additional gradient estimation errors it entails tend to largely offset this recovery such that the net error made unwrapping a given terrain (i.e. at fixed h instead of fixed wrong estimation ratio) seems to be significantly worse the higher the redundancy, except for cases involving flat surfaces plus Gaussian noise or a ramp with atmospheric noise. Also, it seems like the information recovered via the redundancy at a given wrong estimations ratio is only significant for very small r and when the profile is straightforward (e.g. a toy model). It becomes insignificant when a painting or real data is involved. There are still critical thresholds when dealing with nonzero redundancy and plotting as a function of the wrong estimation ratio, but it is very often lower than the one with no redundancy.

The takeaway is that perfect unwrapping is only achievable when a reasonably small number of gradients are wrongly estimated. Typically, this means that no more than 5% (and preferably fewer than 3%) of the gradients should be greater than h in absolute value (the more uneven the terrain, the smaller the threshold). Also, it is almost always better not to use any redundancy at all, except when one is dealing with atmospheric noise. Not only does the MCF algorithm run much faster than

LP, but it is also more precise in the unwrapping process.

4.8.2 On generic gradient correcting

Closely related to phase unwrapping, the more general method involving the determination of the closest gradient in L_1 norm to an approximate, noisy gradient was studied. Contrary to phase unwrapping with enforced congruency, redundancy is much more helpful than none and reduces noise very significantly even when noise levels are high, involving both global centered Gaussian noise and severe but local and sparse impulse noise.

In the case when the δ are not constrained to be congruent, LP with redundancy $r > 0$ outperforms MCF and LP with $r = 0$ by a wide margin, and increasing the redundancy further improves the image or field recovery. In other words, the behavior of the recovery (which, in this case, is an integration process rather than an unwrapping process) is opposite to that of conventional unwrapping trying to invert a modulo operator: the more redundancy, the better in almost all cases, even when the noise level is very high. This might be due to the fact that for fixed values of reasonably low probability of an arc incurring severe impulse noise, adding a new gradient implies information gain very often, whereas in the modular noise, adding a gradient longer than usual often implies adding an error due to a large gradient (larger than half the modulus), resulting in information loss, thereby explaining the apparently contradictory phenomena.

5 Conclusion

We have implemented Costantini's algorithms to unwrap data with a minimum-cost flow method with no redundancy and with linear programming with redundancy in a general setting (Delaunay triangulation, possibly redundant). We have investigated the effects of redundancy on the unwrapping process thoroughly and found a few (but not none) cases in which redundancy is helpful (e.g. a relief map plus severe atmospheric noise). However, in most cases, the increase in redundancy proves to be deleterious for the unwrapping process and the algorithm, requiring linear programming instead of minimum-cost flow, is much slower. Therefore in most cases, it is reasonable to use no redundancy at all unless one is dealing with a specific case of noise (either Gaussian noise on a flat region or atmospheric noise on a gentle slope). It is also worth mentioning that in [5], the author uses linear programming to unwrap a graph constructed using the Persistent Scatterer Pairs (PSP) [4] technique. While the aforementioned graph was shown to yield better connections than a simple Delaunay triangulation, it is not planar in the general case. Therefore, the use of linear programming is necessary for this graph.

In the case of generic gradient correction in L_1 norm when trying to recover a field on a graph with a noisy, approximate gradient involving global, low-intensity centered Gaussian noise and high-intensity but sparse and local impulse noise, using redundancy aids significantly the recovery, and using no redundancy at all usually yields lower-quality results. The behavior of the recovery with the redundancy seems almost opposite to what happens in conventional unwrapping with enforced congruency. This might be due to the fact that in regular noise, adding a gradient often conveys an information gain whereas in the modular noise, adding a gradient, which links points farther away from one another, usually results in a mistake and information loss. In the case of PS deformation velocity and residual topography estimation, the connection of spatially distant points through re-

dundancy might decrease the quality of the estimate due to the increase in atmospheric noise and nonlinear deformation phenomena. In this case, the previous recommendation of avoiding large redundancy might still be valid.

All in all, it seems like redundancy helps denoising regular noise (Gaussian, impulse, atmospheric...) rather than modular, "wrapping" noise. This might be due to the fact that in regular noise, new gradients often convey information due to the low probability of incurring a severe impulse noise, whereas in modular noise, they often lead to information loss because points far from one another lead to large gradients which, as we have seen, necessarily induce an error when the gradient is larger than half the modulus.

6 Demo interface

The default mode showcases phase unwrapping. The user can either upload an image or use one of the several blobs available and wrap it modulo $2h$ where $h > 0$ is at the user's discretion. The user can choose MCF over LP when the redundancy level is set to 0 by checking the corresponding checkbox, otherwise LP is used. The user can also choose the "small basis" option to prompt the algorithm to find a small cycle basis as described in section 3.2, otherwise it uses a fundamental cycle basis. The redundancy level is also left at the user's discretion, however its maximum value cannot exceed 2 because otherwise the computing times would be forbiddingly long by IPOL's standards even for small images. An example of such a mode is shown in Figure 35, along with an execution result using the Mona Lisa blob and running the algorithm with the default parameters in Figure 36. In the latter figure, the "compare" checkbox was checked, the "input" chosen on the left and corresponding to the wrapped initial input image, and the "output" on the right where an unwrapping attempt was made. The quality of the unwrapping can only be measured against the initial ground truth provided by the user which is not shown, and not the wrapped input which is shown only as a reference for the user to witness the uncanny nature of the noise induced by the wrapping process and its intensity. At the bottom, in the gray rectangular area, information about the time taken to process the image is indicated, and at the very bottom left of the screen, an option to download the output in the form of a .tif file is provided.

An integration mode is also available via a radio button. When this option is chosen, all the previous options remain valid except for half-modulus h which is not used anymore as this button implements the integration technique of 3.4 where instead of wrapping the image, a noise whose intensity and nature at the user's discretion is added to its gradients and a gradient-correcting attempt is made in L_1 norm. From this attempt, a signal reconstruction is done up to a global constant. The option to use a priori knowledge of the noise levels is also left to the user to aid the denoising process in the form of an additional checkbox. An example of such a mode is shown in Figure 37, along with an execution result using the Mona Lisa blob and running the algorithm with the default parameters in Figure 38. In the latter figure, the "compare" checkbox was checked, the "input" chosen on the left and corresponding to the initial input image (since the noise is only added to the gradients and not to the image directly, it is not possible to show a noisy input version like in the unwrapping mode), and the "output" on the right where a gradient-correcting attempt was made, followed by an integration. Contrary to the unwrapping mode, the quality of the correction can be witnessed directly by comparing the input with the output. Just like in the unwrapping mode, the information about the running time and an opportunity to download the output is provided at the bottom.

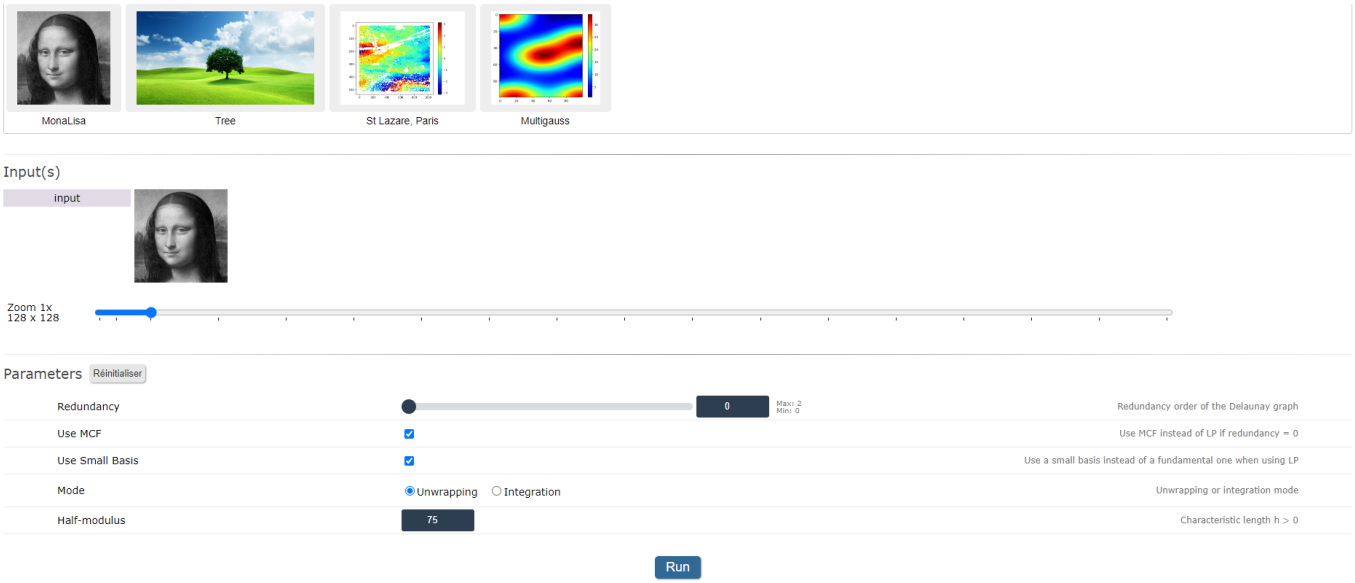


Figure 35: Demo interface in Unwrapping mode

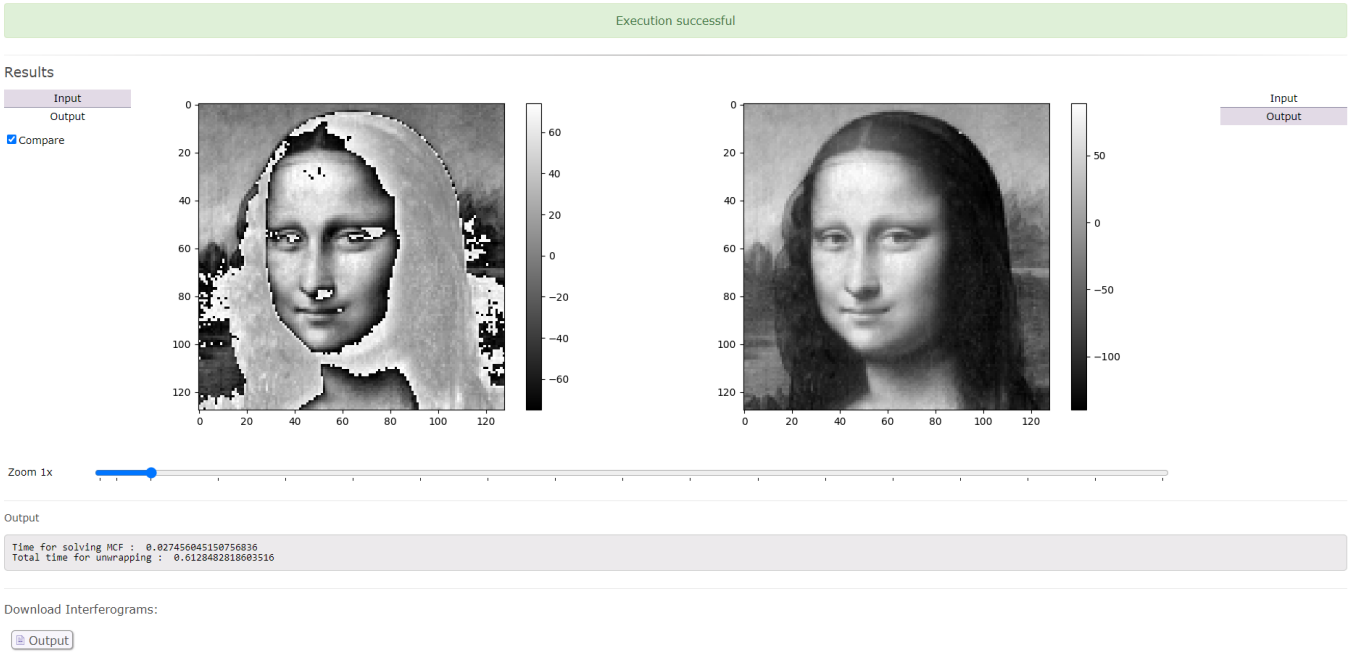


Figure 36: Default execution of phase unwrapping for Mona Lisa

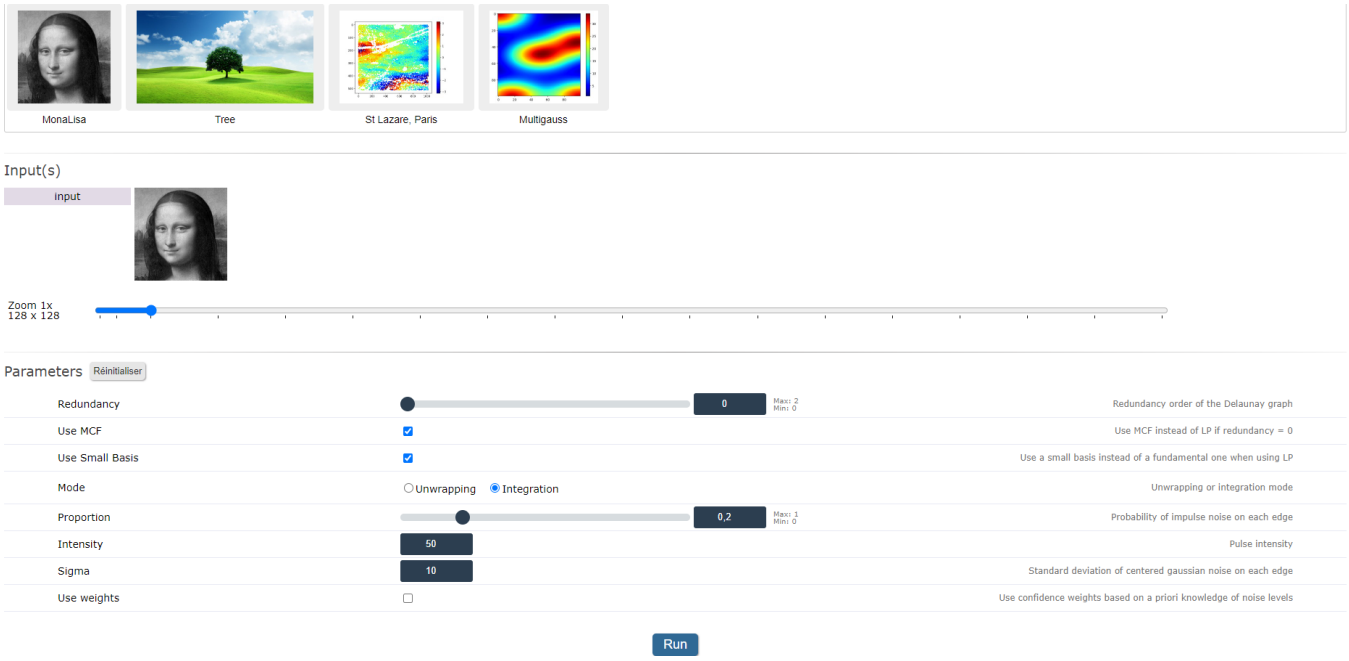


Figure 37: Demo interface in Integration mode

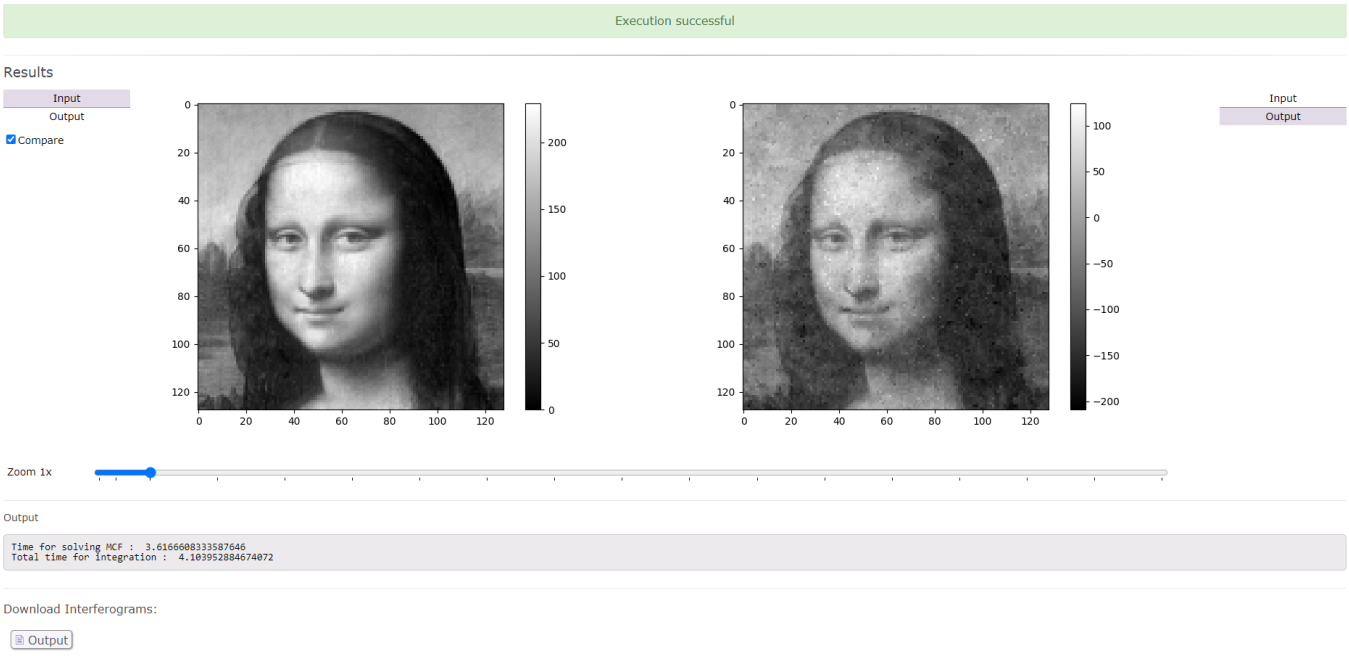


Figure 38: Default execution of Integration for Mona Lisa

A A bound on residues

This appendix is dedicated to the proof of the following statement:

Proposition 3. *Integrating the wrapped gradient of a wrapped signal modulo 2π around a cycle C of length n yields a residue bounded in absolute value satisfying*

$$\left|\frac{Res_{\,C}}{2\pi}\right| \leq \begin{cases} \frac{n-1}{2} \text{ if } n \text{ is odd} \\ \frac{n}{2}-1 \text{ if } n \text{ is even.} \end{cases}$$

Proof:

Suppose C is a cycle of length $n \geq 2$ whose points are numbered in a certain direction along it from 1 to n and that we have $f'_{i,j} \coloneqq \mathcal{W}(\overline{\varphi_j} - \overline{\varphi_i})$, where the $\overline{\varphi_i}$ are the phases φ_i wrapped into the interval $[-\pi, \pi]$. In particular, we saw that $\overline{\varphi_j} - \overline{\varphi_i} \in (-2\pi, 2\pi)$ so $f'_{i,j} \coloneqq \overline{\varphi_j} - \overline{\varphi_i} + 2\pi a_{i,j}$ where $a_{i,j} \in \{0, \pm 1\}$ is the ambiguity. Therefore, we have

$$\mathrm{Res}_C = f'_{n,1} + \sum_{i=1}^{n-1} f'_{i,i+1} = 2\pi \left(a_{n,1} + \sum_{i=1}^{n-1} a_{i,i+1} \right).$$

Now let $\epsilon > 0$ be small enough, and

$\overline{\varphi_1} = \pi - \epsilon,$
$\overline{\varphi_2} = \overline{\varphi_1} - \pi - \epsilon = -2\epsilon,$
$\overline{\varphi_3} = \overline{\varphi_2} + \pi - \epsilon = \pi - 3\epsilon,$
$\overline{\varphi_4} = \overline{\varphi_3} - \pi - \epsilon = -4\epsilon,$
$\overline{\varphi_5} = \overline{\varphi_4} + \pi - \epsilon = \pi - 5\epsilon,$
$\vdots$
$$\overline{\varphi_n} = \overline{\varphi_{n-1}} - (-1)^n\pi - \epsilon = (n \bmod 2)\pi - n\epsilon = \begin{cases} \pi - n\epsilon \text{ if } n \text{ is odd} \\ -n\epsilon \text{ if } n \text{ is even,} \end{cases}$$

by starting from $\pi - \epsilon$ and alternatively subtracting $\pi + \epsilon$ or adding $\pi - \epsilon$. If ϵ is small enough for fixed n, then the $\overline{\varphi_i}$ all lie in $[-\pi, \pi)$, so this choice is feasible. Obviously, this yields

$a_{1,2} = 1,$
$a_{2,3} = 0,$
$a_{3,4} = 1,$
$a_{4,5} = 0,$
$\vdots$
$$a_{n-1,n} = \begin{cases} 0 \text{ if } n \text{ is odd} \\ 1 \text{ if } n \text{ is even,} \end{cases}$$
$$a_{n,1} = \begin{cases} 0 \text{ if } n \text{ is odd} \\ -1 \text{ if } n \text{ is even.} \end{cases}$$

This feasible choice obviously yields

$$\frac{\mathrm{Res}_C}{2\pi} = \begin{cases} \frac{n-1}{2} \text{ if } n \text{ is odd} \\ \frac{n}{2}-1 \text{ if } n \text{ is even.} \end{cases}$$

It remains to show that this choice maximizes $\frac{\text{Res}_C}{2\pi}$.

To achieve this, consider the list

$$a \coloneqq a_{1,2};\, a_{2,3};\, \ldots a_{n-1,n};\, a_{n,1}$$

as cyclic (i.e. every term has a left and a right neighbor, the left neighbor of the first term is the last term and the right neighbor of the last term is the first one). With this is mind, it is easy to see that no two consecutive 1s can occur, otherwise we would have

$$\overline{\varphi_{i+1}} - \overline{\varphi_i} + \overline{\varphi_i} - \overline{\varphi_{i-1}} = \overline{\varphi_{i+1}} - \overline{\varphi_{i-1}} < -2\pi, i \in \{2, \ldots, n-1\}$$

or

$$\overline{\varphi_1} - \overline{\varphi_n} + \overline{\varphi_2} - \overline{\varphi_1} = \overline{\varphi_2} - \overline{\varphi_n} < -2\pi$$

which is a contradiction.

Therefore, the first $n-1$ values of a, which are $1, 0, 1, 0, \ldots$ maximize the sum of the first $n-1$ possible values. When n is odd, the last value of a is 0. It cannot be equal to 1, for otherwise along with its right neighbor in the cyclic list, i.e. with $a_{1,2}$, they would constitute two consecutive 1s, which has been shown not to be possible. Therefore when n is odd, this feasible choice maximizes $\frac{\text{Res}_C}{2\pi}$. While $0, 1, 0, 1, \ldots$ would also maximize the sum of the first $n-1$ possible values (and would be the only alternative), the same result holds as the last ambiguity would not be equal to 1 since its left neighbor would also be equal to 1.

When n is even, the first $n-1$ values of a still maximize the sum of the first $n-1$ possible values, but the last value of a is -1. It could never have been 1 because its right neighbor $a_{1,2}$ in the cyclic list is equal to 1. Therefore it remains to show that it is not possible to obtain the list $1, 0, 1, 0, \cdots, 1, 0$ when n is even. Indeed, because of the "no two consecutive ones" rule, if the list started with a 0, the $n-1$ first values $(0, 1, 0, \ldots)$ would sum to at most one unit less than the previous values $(1, 0, 1, \ldots)$, so if it could be shown that the last ambiguity is 1 (its maximum value), then the sum of all the ambiguities would not exceed the previous sum with the list starting with 1 and ending with the last value equal to 0. Therefore, it remains to show that the last value $a_{n,1}$ cannot be 0 when the first $n-1$ values are $1, 0, 1, 0, 1, \ldots$ to conclude.

Suppose this were the case. Then the cyclic list would endlessly alternate between 0 and 1 (contrary to the case when n is odd). This would imply

$$0 = (\overline{\varphi_2} - \overline{\varphi_1}) + (\overline{\varphi_3} - \overline{\varphi_2}) + (\overline{\varphi_4} - \overline{\varphi_3}) + \ldots + (\overline{\varphi_n} - \overline{\varphi_{n-1}}) + (\overline{\varphi_1} - \overline{\varphi_n}) < \frac{n}{2}\pi - \frac{n}{2}\pi = 0$$

which is a contradiction.

A similar reasoning could have been done, replacing the 1s with -1s, so the bound holds in absolute value.

This is enough to conclude. □

Acknowledgment

I am very grateful and would like to thank the people involved in:

- **Kayrros** for participating in the funding of this project.
- **OR-Tools open-source software** used to implement MCF. [20]
- **PuLP open-source software** used to implement LP. [21]
- **SciPy open-source software** underpinning the implementation of all the algorithms. [23]

Image Credits

 Leonardo da Vinci, Public domain, via Wikimedia Commons

References

[1] Dimitri Bertsekas, *Network optimization: continuous and discrete models*, vol. 8, Athena Scientific, 1998.

[2] Curtis W. Chen and Howard A. Zebker, *Phase unwrapping for large sar interferograms: Statistical segmentation and generalized network models*, IEEE Transactions on Geoscience and Remote Sensing, 40 (2002), pp. 1709–1719.

[3] Mario Costantini, *A novel phase unwrapping method based on network programming*, IEEE Transactions on Geoscience and Remote Sensing, 36 (1998), pp. 813–821.

[4] Mario Costantini, Salvatore Falco, Fabio Malvarosa, and Federico Minati, *A new method for identification and analysis of persistent scatterers in series of sar images*, International Geoscience and Remote Sensing Symposium (IGARSS), 2 (2008), pp. 449–452.

[5] Mario Costantini, Fabio Malvarosa, and Federico Minati, *A general formulation for redundant integration of finite differences and phase unwrapping on a sparse multidimensional domain*, IEEE Transactions on Geoscience and Remote Sensing, 50 (2012), pp. 758–768.

[6] Benjamin Dubois-Taine, Roland Akiki, and Alexandre D'Aspremont, *Iteratively Reweighted Least Squares for Phase Unwrapping*, jan 2024.

[7] M. Eineder and J. Holzner, *Phase unwrapping of low coherence differential interferograms*, International Geoscience and Remote Sensing Symposium (IGARSS), 3 (1999), pp. 1727–1730.

[8] Alessandro Ferretti, Claudio Prati, and Fabio Rocca, *Nonlinear subsidence rate estimation using permanent scatterers in differential SAR interferometry*, IEEE Transactions on Geoscience and Remote Sensing, 38 (2000), pp. 2202–2212.

[9] ——, *Permanent scatterers in SAR interferometry*, IEEE Transactions on Geoscience and Remote Sensing, 39 (2001), pp. 8–20.

[10] Dennis C Ghiglia and Louis A Romero, *Minimum lp-norm two-dimensional phase unwrapping*, Journal of the Optical Society of America A, 13 (1996), p. 1999.

[11] Richard M. Goldstein, Howard A. Zebker, and Charles L. Werner, *Satellite radar interferometry: Two-dimensional phase unwrapping*, Radio Science, 23 (1988), pp. 713–720.

[12] Michel Gondran and Michel Minoux, *Graphes et algorithmes (4e ed.)*, Lavoisier, 2009.

[13] BM Kampes, RF Hanssen, and Z Perski, *Radar interferometry with public domain tools*, in FRINGE 2003 Workshop, vol. 550, 2004.

[14] Bert M. Kampes, *Radar interferometry: Persistent scatterer technique*, vol. 12, 2006.

[15] Casimir Kuratowski, *Sur le problème des courbes gauches en topologie*, Fundamenta Mathematicae, 15 (1930), pp. 271–283.

[16] Tian Lan, Deniz Erdogmus, Susan J. Hayflick, and J. Urick Szumowski, *Phase unwrapping and background correction in mri*, in 2008 IEEE Workshop on Machine Learning for Signal Processing, 2008, pp. 239–243.

[17] Saunders Mac Lane, *A combinatorial condition for planar graphs*, Fundamenta Mathematicae, 28 (1937), pp. 22–32.

[18] Christian Liebchen and Romeo Rizzi, *Classes of cycle bases*, Discrete Applied Mathematics, 155 (2007), pp. 337–355.

[19] Keith Paton, *An algorithm for finding a fundamental set of cycles of a graph*, Communications of the ACM, 12 (1969), pp. 514–518.

[20] Laurent Perron and Vincent Furnon, *Or-tools*.

[21] J.S. Roy, S.A. Mitchell, and F. Peschiera, *Pulp*.

[22] S. K. Stein, *Convex maps*, Proceedings of the American Mathematical Society, 2 (1951), pp. 464–466.

[23] Pauli Virtanen, Ralf Gommers, Travis E. Oliphant, Matt Haberland, Tyler Reddy, David Cournapeau, Evgeni Burovski, Pearu Peterson, Warren Weckesser, Jonathan Bright, Stéfan J. van der Walt, Matthew Brett, Joshua Wilson, K. Jarrod Millman, Nikolay Mayorov, Andrew R. J. Nelson, Eric Jones, Robert Kern, Eric Larson, C J Carey, İlhan Polat, Yu Feng, Eric W. Moore, Jake VanderPlas, Denis Laxalde, Josef Perktold, Robert Cimrman, Ian Henriksen, E. A. Quintero, Charles R. Harris, Anne M. Archibald, Antônio H. Ribeiro, Fabian Pedregosa, Paul van Mulbregt, and SciPy 1.0 Contributors, *SciPy 1.0: Fundamental Algorithms for Scientific Computing in Python*, Nature Methods, 17 (2020), pp. 261–272.

[24] Kui Zhang, Di Wu, Shu Wang, Ruiqing Song, and Guojie Meng, *A new method for estimating unambiguous phase observations of re-identified coherent targets for multi-baseline insar techniques*, Remote Sensing Letters, 8 (2017), pp. 1172–1179.