A rigorous numerical algorithm for computing the linking number of links

Zin Arai

Department of Mathematics, Hokkaido University
zin@math.sci.hokudai.ac.jp

January 31, 2013

Abstract

We propose a rigorous numerical algorithm for computing the linking number of links defined by spatially distributed data points. The key idea is to use an analytic expression for the solid angle of a tetrahedron for quick evaluation of the degree of the Gauss map. An implementation of the algorithm with INTLAB, a Matlab toolbox for reliable computing, is also provided.

1 linking number

Recently we have seen an increasing interest in computational topology. More and more applications appear inside and outside mathematics: image analysis, sensor networks, protein/cell structures, etc. The main tool of computational topology have been, so far, the set of algorithms to compute the homology group of given geometric data, such as CHomP [1]. The aim of the paper is to add a new tool to computational topology by developing a rigorous algorithm for computing the linking number of links. Note that the linking number is a homotopical, rather than homological, invariant.

Our algorithm is designed to handle spatially distributed data points directly; it do not assume that the projection of the link to a plane is known. This is because what we have in mind as input data is the data coming from numerical or physical experiments, and thus taking projection of such noisy experimental data might cause numerical unstability. The output of CHomP also fits into this format and therefore we can easily combine CHomP and our algorithm.

We begin by recalling the definition of the linking number. The linking number $lk(K, L)$ of two disjoint oriented knots $K, L : S^1 \to \mathbb{R}^3$ is a topological invariant which counts the number of times that each curve winds around the other. The precise definition of the linking number can be given in several

equivalent ways [3]; the one we will apply involves the Gauss map f from $T^2 = S^1 \times S^1$ to S^2 defined by

$$f(u,v) = \frac{K(u) - L(v)}{|K(u) - L(v)|},$$

which is well defined since we assume that K and L are disjoint. Then $lk(K,L)$ is defined to be the degree $\deg(f)$ of the Gauss map, namely the integer which satisfies

$$f_*([T^2]) = \deg(f)[S^2].$$

Here $[T^2] \in H_2(T^2) \cong \mathbb{Z}$ and $[S^2] \in H_2(S^2) \cong \mathbb{Z}$ are the fundamental classes of T^2 and S^2, respectively. In our case of $f : S^1 \times S^1 \to S^2$, this number coincides with the *oriented* area of the image of f divided by the area of S^2, that is,

$$lk(K,L) = \deg(f) = \frac{\mathrm{vol}\,(f(T^2))}{\mathrm{vol}\,(S^2)} \tag{1}$$

where "vol" denotes the area on the sphere S^2.

In the following, we assume that K and L are represented by points distributed in $\mathbb{R}^3$. Precisely, the data we are given is two sets of points $\{k_1, k_2, \dots k_m\}$ and $\{l_1, l_2, \dots l_n\} \subset \mathbb{R}^3$. The knots K and L are defined to be the piecewise-affine closed loop connecting these points in the cyclic order. Remark that since the linking number is a homotopy invariant, we can also handle smooth knots within this framework by taking piecewise-linear approximations of the knots.

Then, the area we want to compute is decomposed into a sum of the areas of smaller regions on the torus as follows:

$$\mathrm{vol}\,(f(T^2)) = \sum_{1 \le i \le m} \sum_{1 \le j \le n} \mathrm{vol}\,(f(T_{ij})), \tag{2}$$

where T_{ij} is a rectangular region on T^2 defined by four points

$$(p_i, q_j),\ (p_i, q_{j+1}),\ (p_{i+1}, q_j),\ (p_{i+1}, q_{j+1}) \in T^2$$

where $K(p_i) = k_i$, $K(p_{i+1}) = k_{i+1}$, $L(q_j) = l_j$ and $L(q_{j+1}) = l_{j+1}$. Here we identify $m+1$ with 1, and $n+1$ with 1.

Then we ask what the image $f(T_{ij})$ is. First we focus on a boundary segment of T_{ij} that connects (p_i, q_j) and (p_i, q_{j+1}). Since we assume that K and L are piecewise-affine curves, the image of this segment is a piece of great circle passing through two points on the unit sphere that are given as the intersections of the sphere with the lines connecting the origin to $l_j - k_i$ and $l_{j+1} - k_i$. The image of other boundary segments can similarly be described as great circle arcs.

To compute each summand $\mathrm{vol}\,(f(T_{ij}))$, we will make use of an equality obtained by Van Oosterom and Strackee [2]. Consider a tetrahedron based at the origin of $\mathbb{R}^3$ and spanned by three non-zero vectors a, b and $c \in \mathbb{R}^3$. Then the equality expresses the solid angle $\Omega(a,b,c)$ of the tetrahedron in terms of these vectors:

$$\tan\left(\frac{1}{2}\Omega(a,b,c)\right) = \frac{[a\ b\ c]}{|a||b||c| + (a \cdot b)|c| + (c \cdot a)|b| + (b \cdot c)|a|}.$$

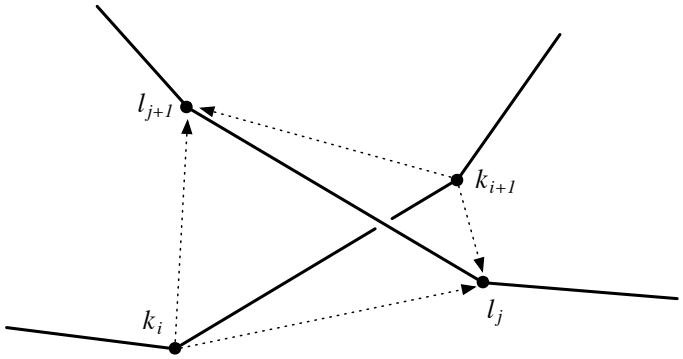

Figure 1: Pieces of K, L and the vectors connecting vertices.

Here $[a\ b\ c]$ denotes the triple scalar product $a \cdot (b \times c)$ of a, b and c (coincides with the determinant of the 3-by-3 matrix composed of a, b and c). Remark that even when vectors a, b and c are dependent, the equality holds because then the tetrahedron is collapsed and thus its solid angle is 0, while $[a\ b\ c]$ is also 0. We notice that $\Omega(a, b, c)$ is the area of the triangle on the unit sphere whose vertices are the intersection of lines connecting the origin with a, b and c (see Figure 2).

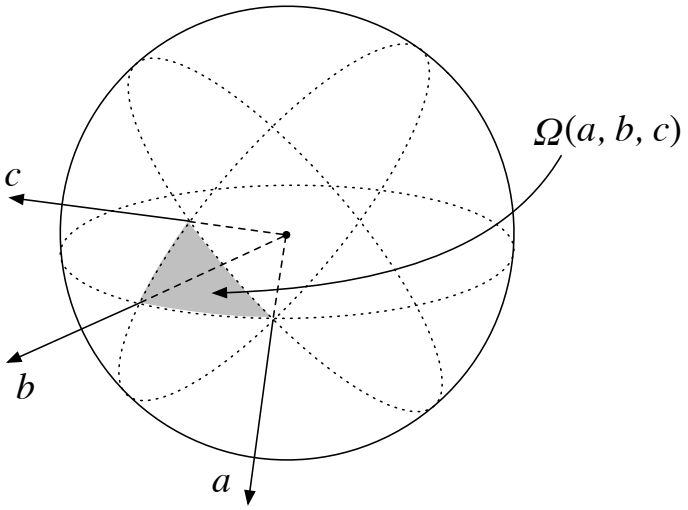

Figure 2: The solid angle of the spherical triangle defined by a, b and c.

Therefore, by applying the arc-tangent function, we obtain

$$\begin{aligned}\frac{1}{2}\operatorname{vol}\left(f(T_{ij})\right) = &\arctan\left(\frac{[\alpha\ \beta\ \gamma]}{|\alpha||\beta||\gamma| + (\alpha\cdot\beta)|\gamma| + (\gamma\cdot\alpha)|\beta| + (\beta\cdot\gamma)|\alpha|}\right)\\ &+\arctan\left(\frac{[\gamma\ \delta\ \alpha]}{|\gamma||\delta||\alpha| + (\gamma\cdot\delta)|\alpha| + (\alpha\cdot\gamma)|\delta| + (\delta\cdot\alpha)|\gamma|}\right)\end{aligned} \tag{3}$$

where

$$\alpha = l_j - k_i, \quad \beta = l_j - k_{i+1}, \quad \gamma = l_{j+1} - k_{i+1}, \quad \delta = l_{j+1} - k_i.$$

Here by $\arctan(y/x)$ we precisely mean `atan2(y, x)`, the angle in radians between the positive x-axis of a plane and the point (x, y) (see [2] for the detail).

Summarizing, we can compute the linking number $lk(K, L)$ by substituting the equality (3) into (2) and then (2) into (1).

2 Algorithm and Implementation

In the previous section, we saw that the degree of the Gauss map can be evaluated by computing the following objects: the inner product, the norm and the scalar product of vectors; the arc-tangent of a real number. All these computations can be manipulated on a computer, in a mathematically rigorous way, by introducing interval arithmetic.

The following is a straightforward implementation of the formula given in the previous section.

Algorithm 1 (Computing the Linking Number)
Input: floating-point vectors $\{k_1, k_2, \dots k_m\}$, $\{l_1, l_2, \dots l_n\} \subset \mathbb{R}^3$.
Output: an interval I

```
interval I = [0, 0]
interval_vectors α, β, γ, δ
for i = 1 to m do
  for j = 1 to n do
    α = l_j − k_i,   β = l_j − k_{i+1},   γ = l_{j+1} − k_{i+1},   δ = l_{j+1} − k_i.
    I = I + Solid_Angle(α, β, γ) + Solid_Angle(γ, δ, α)
  end for
end for
I = I/(4π)
return I
```

Here by `Solid_Angle` we mean a function that takes three interval vectors a, b, c and returns an interval containing the solid angle $\Omega(a, b, c)$. The function `Solid_Angle` may return an error, as we will see in Algorithm 3, and if this is the case then Algorithm 1 terminates with an error.

Provided that all the computations are executed rigorously without any error, the true value of $\operatorname{vol}(\operatorname{Im}(f))$ and thus the linking number $lk(K, L)$ should be contained in the resulting interval I. Therefore, we have the following theorem.

Theorem 2 *If Algorithm 1 returns an interval that contains exactly one integer l, then we have $l = lk(J, K)$.*

The algorithm above is implemented with INTLAB [4], a Matlab toolbox for interval arithmetic, and the source code is available at the author's web site `http://www.math.sci.hokudai.ac.jp/~zin/`

There is no essential difficulty in implementing the algorithm with INTLAB. However, we need to pay some attention to the discontinuity of `atan2(y, x)`, which appears on $\{x < 0,\ y = 0\}$. Since INTLAB does not have `atan2` function, our implementation of `solid_angle` use `atan` function instead as follows.

Algorithm 3 (Computing the Solid Angle with `atan`)
Input: `interval_vectors` a, b, c
Output: `interval` A = `Solid_Angle`(a, b, c)

```
interval I, J
J = |a||b||c| + (a·b)|c| + (c·a)|b| + (b·c)|a|
if (0 ∈ J) then
  if (I > 0) then
    A = −atan(J/I) + π/2
  else if (I < 0) then
    A = −atan(J/I) − π/2
  else
    return error
  end if
else
  I = [a b c]
  if (J > 0) then
    A = atan(I/J)
  else if (I > 0) then
    A = atan(I/J) + π
  else if (I < 0) then
    A = atan(I/J) − π
  else
    return error
  end if
  return 2 ∗ A
end if
```

Here by $I > 0$ we mean that both ends of the interval I is greater than 0. The Algorithm 3 fails when $I = [abc]$ and $J = |a||b||c| + (a \cdot b)|c| + (c \cdot a)|b| + (b \cdot c)|a|$ both contain 0. It also fails when the discontinuity of `atan2` appears, that is, when J is negative and I contains 0. In practical applications, as we will see in the next section, these error rarely occur.

Remark 4 Assume that we do not know the exact value of $\{k_1, k_2, \ldots k_m\}$ and $\{l_1, l_2, \ldots l_n\}$, but we are given the sets $\{K_1, K_2, \ldots K_m\}$ and $\{L_1, L_2, \ldots L_n\}$

where $K_i, L_j \subset \mathbb{R}^3$ are products of intervals and such that $k_i \in K_i$ and $l_j \in L_j$ hold. The set K_i can be regarded as an "error bar" around the point k_i. Our implementation can equally be applied to this noisy data without any change since INTLAB library can handle numbers and intervals in the same manner; we just input K_i and L_j (`interval_vector`) instead of k_i and l_j (`vector`). As we will see in the next section, This simple treatment of noise, however, works fine only when the amplitude of the noise is small enough. See Example 7 in the next section.

3 Examples

In this section, we show some examples of the application of the algorithm. All the computation is done on an Apple MacPro with 2.93 GHz Intel Xeon CPU.

Example 5 (Hopf link) The Hopf link is the simplest non-trivial link consisting two unknotted circles linked together so that the linking number is ± 1, depending on the orientations. Consider two circles K and L represented by

$$\{k_1, k_2, \ldots, k_8\} = \left\{ \begin{pmatrix}1\\0\\0\end{pmatrix}, \begin{pmatrix}1\\1\\0\end{pmatrix}, \begin{pmatrix}0\\1\\0\end{pmatrix}, \begin{pmatrix}-1\\1\\0\end{pmatrix}, \begin{pmatrix}-1\\0\\0\end{pmatrix}, \begin{pmatrix}-1\\-1\\0\end{pmatrix}, \begin{pmatrix}0\\-1\\0\end{pmatrix}, \begin{pmatrix}1\\-1\\0\end{pmatrix} \right\}$$

and

$$\{l_1, l_2, \ldots, l_8\} = \left\{ \begin{pmatrix}0\\0\\0\end{pmatrix}, \begin{pmatrix}0\\0\\1\end{pmatrix}, \begin{pmatrix}1\\0\\1\end{pmatrix}, \begin{pmatrix}2\\0\\1\end{pmatrix}, \begin{pmatrix}2\\0\\0\end{pmatrix}, \begin{pmatrix}2\\0\\-1\end{pmatrix}, \begin{pmatrix}1\\0\\-1\end{pmatrix}, \begin{pmatrix}0\\0\\-1\end{pmatrix} \right\}.$$

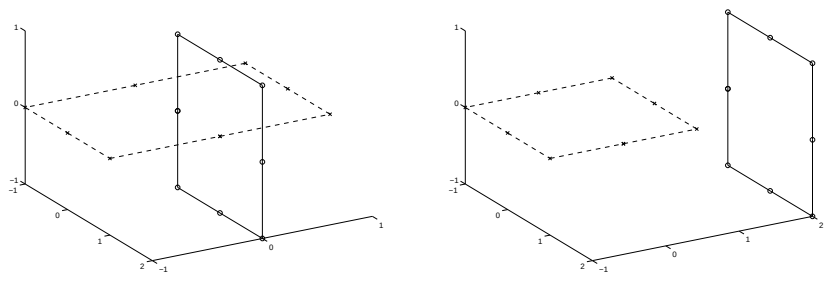

Figure 3: The Hopf link (left) and a trivial link (right).

Topologically, they form a Hopf link whose liking number is $+1$, as illustrated in Figure 3. The dotted line and the solid line represent the knots K and L, respectively. By applying our algorithm to these points, we obtain an interval $[0.99999999999999, 1.00000000000001]$. It contains exactly one integer, namely the correct linking number $lk(K, L) = 1$.

Example 6 (a trivial link) Now we replace L of the previous example with a circle given by

$$\{l_1, l_2, \ldots, l_8\} = \left\{ \begin{pmatrix} 0 \\ 2 \\ 0 \end{pmatrix}, \begin{pmatrix} 0 \\ 2 \\ 1 \end{pmatrix}, \begin{pmatrix} 1 \\ 2 \\ 1 \end{pmatrix}, \begin{pmatrix} 2 \\ 2 \\ 1 \end{pmatrix}, \begin{pmatrix} 2 \\ 2 \\ 0 \end{pmatrix}, \begin{pmatrix} 2 \\ 2 \\ -1 \end{pmatrix}, \begin{pmatrix} 1 \\ 2 \\ -1 \end{pmatrix}, \begin{pmatrix} 0 \\ 2 \\ -1 \end{pmatrix} \right\}.$$

Then K and L are not linked to each other and therefore the linking number should be 0 (see Figure 3). Actually, the algorithm returns $[-0.37437779063061 \times 10^{-15}, 0.47156140589755 \times 10^{-15}]$, which contains exactly one integer 0, which is the correct linking number.

Example 7 (Hopf link with noise) We again consider the Hopf link defined by the same points as Example 5, but now we put a bounded noise on data points. Namely, each data point is replaced by a regular hexahedron centered at the point. For example, if we replace each point with a regular hexahedron with radius 0.02, then our algorithm returns $[0.32445624257739, 1.80336874019784]$. This interval still contains only one integer and therefore we can conclude any link that is given by points contained in these regular hexahedrons should have the same linking number, 1. However, if we use the hexahedron of radius 0.03, then the resulting interval is $[-0.01296718559705, 2.31661672274700]$, which contains $0, 1$ and 2. In this case thus we can not determine the linking number uniquely.

Example 8 (randomly generated links) Now we examine the validity of the algorithm with more practical examples involving a larger number of points that are distributed in a complicated manner. For this purpose, we consider the knots K and L defined by randomly generated points in $[0, 1]^3 \subset \mathbb{R}^3$. Figure 4 shows an example of such a link with $n = m = 20$ data points for which our algorithm returns $[2.99999999999976, 3.00000000000024]$. The only integer contained in this interval is 3, therefore we can conclude the linking number $lk(K, L)$ is 3.

The next example, shown in Figure 5, is a randomly generated link with $n = 10$ and $m = 40$. Note that the product $n \times m$ is the same as before. For this link, our algorithm returns $[-2.00000000000021, -1.99999999999979]$, thus we know that the linking number is -2.

Similarly, we obtain $[1.99999999999427, 2.00000000000573]$ for the link with $n = m = 100$ data points shown in Figure 6. Again, there is exactly one integer in this interval, implying that the linking number is 2.

We remark that in all examples above, `solid_angle` does not return any error.

The elapsed times in these examples are 5.996162 seconds for $n = m = 20$, 6.029830 seconds for $n = 10, m = 40$ and 149.592007 seconds for $n = m = 100$. It is easy to see that the computational cost of Algorithm 1 is proportional to $n \times m$. In fact, the ratio of elapsed times $149.592007/5.996162 = 24.833333\ldots$ is almost the same as the expected ratio $100^2/20^2 = 25$.

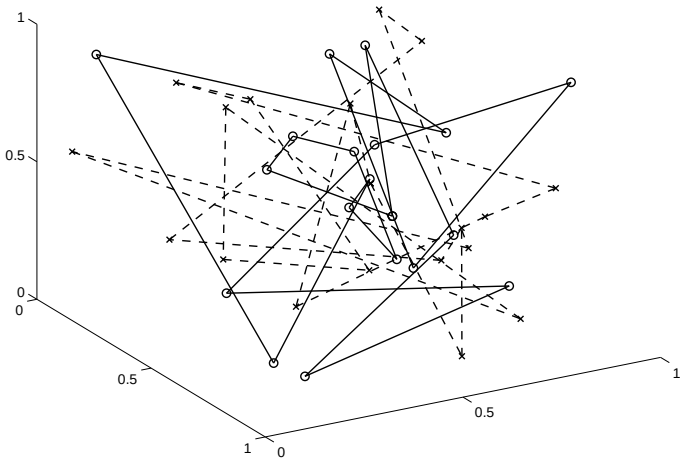

Figure 4: Randomly generated links of length $n = m = 20$.

Acknowledgments

This work is partially supported by JST CREST, and by JSPS Grants-in-Aid for Scientific Research (23684002). The author would like to thank Akane Kawaharada for her helpful comments on the manuscript. Special thanks also go to the anonymous reviewers and Dr. Simon Copar for their helpful remarks that improved the paper.

References

[1] Computational Homology Project (CHomP) website, `http://chomp.rutgers.edu/`

[2] A. Van Oosterom and J. Strackee, "The Solid Angle of a Plane Triangle", *IEEE Trans. Biomed. Eng.*, vol. BME-30, no. 2, pp. 125–126, 1983.

[3] D. Rolfsen, *Knots and Links*, Publish or Perish, 1976.

[4] S. M. Rump, "INTLAB - INTerval LABoratory", in *Developments in Reliable Computing*, ed. T. Csendes, 77–104, Kluwer Academic Publishers, 1999.

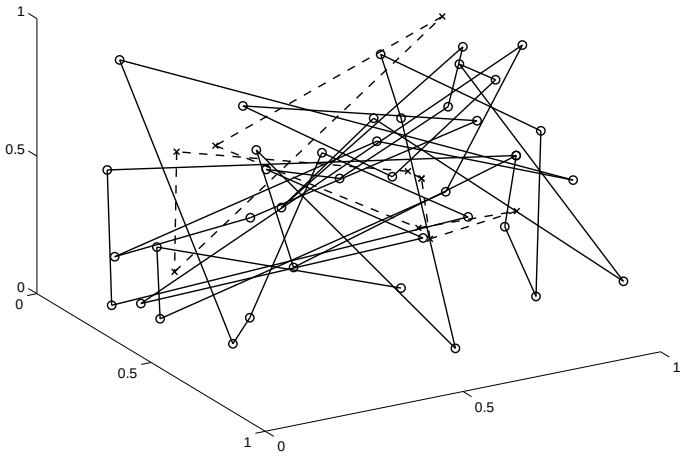

Figure 5: Randomly generated links of length $n = 10, m = 40$.

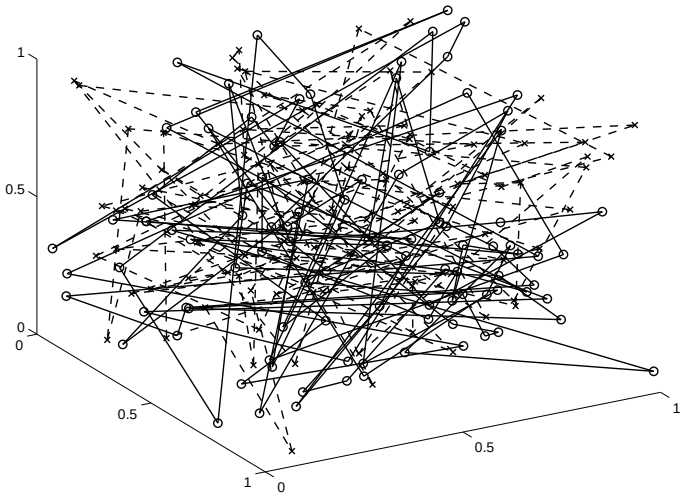

Figure 6: Randomly generated links of length $n = m = 100$.