Modeling of Interpolating Implicit Surfaces

Yifeng Jiang

Advisor: Greg Turk (turk@cc.gatech.edu)

1 Introduction

Given a set of pre-specified points in 2D or 3D space, we are interested in fitting a smooth curve or surface that passes through these constraint points. Smooth interpolation is important in computer-aided design as it allows designers to arbitrarily vary the desired shapes while maintaining good engineering and manufacturing properties induced by smoothness. Computer graphics and computer vision communities also favor smoothness for more appealing visual appearance.

There have been a great amount of work on smooth interpolation and surface reconstruction, since there is no single algorithm that works best in all scenarios. This mini-project starts from the classical approach of thin-plate interpolation [1]. Thin-plate interpolation finds a function $f : \mathbb{R}^2 \to \mathbb{R}$ such that it exactly satisfies the N interpolation points given:

$$f(x_i, y_i) = z_i \quad i = 1, 2, \cdots, N, \tag{1}$$

meanwhile minimizing the total curvature measure of f:

$$E(f) = \int\int_{\mathbb{R}^2} f_{xx}^2(x, y) + 2f_{xy}^2(x, y) + f_{yy}^2(x, y)\, dxdy. \tag{2}$$

It is proved [2] that there is a unique minimizer f^* to this problem:

$$f^*(x, y) = \sum_{i=1}^{N} w_i \phi((x, y) - (x_i, y_i)) + c_0 + c_1 x + c_2 y, \tag{3}$$

where $\phi(x, y) = \|(x, y)\|^2 \log \|(x, y)\|$ is the well-known thin-plate radial basis function, and the weights w_i's must satisfy the following constraints:

$$\sum_{i=1}^{N} w_i = \sum_{i=1}^{N} w_i x_i = \sum_{i=1}^{N} w_i y_i = 0. \tag{4}$$

This unique f^* can thus be obtained by solving a square linear system of equations with respect to the $N + 3$ unknowns $(c_0, c_1, c_2, w_1, \cdots, w_N)$:

$$\begin{bmatrix} 0 & 0 & 0 & 1 & \dots & 1 \\ 0 & 0 & 0 & x_1 & \dots & x_N \\ 0 & 0 & 0 & y_1 & \dots & y_N \\ 1 & x_1 & y_1 & \phi_{11} & \dots & \phi_{1N} \\ \vdots & \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & x_N & y_N & \phi_{N1} & \dots & \phi_{NN} \end{bmatrix} \begin{bmatrix} c_0 \\ c_1 \\ c_2 \\ w_1 \\ \vdots \\ w_N \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 0 \\ z_1 \\ \vdots \\ z_N \end{bmatrix}, \tag{5}$$

where $\phi_{ij} = \phi((x_i, y_i) - (x_j, y_j))$ are known constants.

Two techniques exist for applying interpolation algorithms to modeling curves and surfaces. Given a set of 3D points (x_i, y_i, z_i), suppose we seek a surface interpolating them. The explicit formulation [1] directly fits a $\mathbb{R}^2 \to \mathbb{R}$ function f so that $f(x_i, y_i) = z_i$. This formulation is straightforward to understand since f itself is the desired surface, but it lacks the capability to represent closed surfaces in 3D. The implicit formulation [3] solves this issue by instead fitting a $\mathbb{R}^3 \to \mathbb{R}$ function f so that $f(x_i, y_i, z_i) = 0$. As such, the desired surface is obtained by finding the zero level-set of f, which is one-dimension lower than f. To ensure the surface appears "closed",

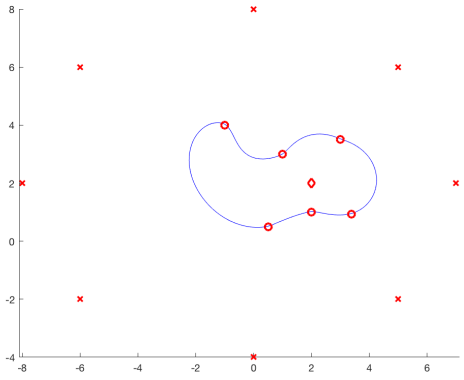


Figure 1: Interpolating a closed curve. The curve needs to pass through the design points drawn in circles. We add some additional constraint points to indicate curve interior and exterior.

we often time add an additional constraint point $f(x_j, y_j, z_j) = 1$ where (x_j, y_j, z_j) is arbitrarily chosen inside the surface; or we add several additional constraint points $f(x_k, y_k, z_k) = -1$ where (x_k, y_k, z_k) are arbitrarily chosen outside the surface. The numbers 1 and -1 are relative scales and do not really matter. As a side note, since we are changing the domain of f from $\mathbb{R}^2$ to $\mathbb{R}^3$ in the implicit formulation, the radial basis function will no longer be in the form of $\|\mathbf{x}\|^2 \log \|\mathbf{x}\|$, but the general derivation above still holds.

Fig. 1 illustrates the basic idea using an example of fitting a closed curve in 2D. We run the interpolating algorithm with the constraint that fitted $f : \mathbb{R}^2 \to \mathbb{R}$ must be zero valued at the circle points, 1-valued at the diamond points, and -1-valued at the cross points. Then we utilize level-set finding algorithms, e.g. marching squares, to locate the zero-value set, which is the interpolated curve (Blue).

One problem with classic thin-plate interpolation is that solving the dense system Eq. 5 takes $O(N^3)$ time, which can be too expensive if number of design (constraint) points become high. We therefore propose two simple techniques to accelerate the fitting process in certain cases. The first technique is based on the observation that in many interactive design scenarios, one could want to refine the current fitted surface by adding few new constraint points. In this case, it is not preferable to recompute the whole solution since most of the points are unchanged. The second technique uses a neural network function, which is a universal function approximator, to represent f. Instead of enforcing interpolation accuracy (Eq. 1) and function smoothness (Eq. 2) in the closed-form solution Eq. 3, we formulate both requirements as a loss function and use gradient-based methods to iteratively optimize f. With the mini-batch training widely used in the machine learning community, the total fitting time grows only linearly with respect to N, which better suits large-scale problems.

Both of our new techniques have guaranteed shorter asymptotic running time than the original algorithm, and our first technique always yields the same closed-form solution as the original one. Our second technique uses statistical approximation, yet evaluation shows it can give qualitatively comparable results with the original approach.

2 Block-inversion Technique for Adding New Constraints

Write Eq. 5 short-handed as $\mathbf{A}\mathbf{w} = \mathbf{b}$, where $\mathbf{w} = (c_0, c_1, c_2, w_1, \cdots, w_N)^T$ being the unknowns to solve. Consider the case when we have solved $\mathbf{w} = \mathbf{A}^{-1}\mathbf{b}$ and obtained the interpolating curve, and would like to add M ($M \ll N$) new constraint points to refine its shape. Then the following new linear system needs to be solved:

$$\begin{bmatrix} \mathbf{A} & \mathbf{C}^T \\ \mathbf{C} & \mathbf{D} \end{bmatrix} \begin{bmatrix} \mathbf{w} \\ \overline{\mathbf{w}} \end{bmatrix} = \begin{bmatrix} \mathbf{b} \\ \overline{\mathbf{b}} \end{bmatrix}, \tag{6}$$

where

$$\mathbf{C} = \begin{bmatrix} 1 & x_{N+1} & y_{N+1} & \phi_{N+1,1} & \cdots & \phi_{N+1,N} \\ 1 & x_{N+2} & y_{N+2} & \phi_{N+2,1} & \cdots & \phi_{N+2,N} \\ \vdots & \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & x_{N+M} & y_{N+M} & \phi_{N+M,1} & \cdots & \phi_{N+M,N} \end{bmatrix} \in \mathbb{R}^{M\times(N+3)}, \tag{7}$$

$$\mathbf{D} = \begin{bmatrix} \phi_{N+1,N+1} & \cdots & \phi_{N+1,N+M} \\ \phi_{N+2,N+1} & \cdots & \phi_{N+2,N+M} \\ \vdots & \ddots & \vdots \\ \phi_{N+M,N+1} & \cdots & \phi_{N+M,N+M} \end{bmatrix} \in \mathbb{R}^{M\times M}, \tag{8}$$

$\overline{\mathbf{w}} = (w_{N+1}, \cdots, w_{N+M})^T$, and $\overline{\mathbf{b}} = (z_{N+1}, \cdots, z_{N+M})^T$. Naively solving Eq. 6 still takes $O(N^3)$ since $\overline{\mathbf{w}}$ and $\mathbf{w}$ are coupled in the new equations. However, we can apply the block-inversion theorem:

$$\begin{bmatrix} \mathbf{A} & \mathbf{C}^T \\ \mathbf{C} & \mathbf{D} \end{bmatrix}^{-1} = \begin{bmatrix} \mathbf{A}^{-1} + \mathbf{A}^{-1}\mathbf{C}^T\mathbf{S}^{-1}\mathbf{C}\mathbf{A}^{-1} & -\mathbf{A}^{-1}\mathbf{C}^T\mathbf{S}^{-1} \\ -\mathbf{S}^{-1}\mathbf{C}\mathbf{A}^{-1} & \mathbf{S}^{-1} \end{bmatrix}, \tag{9}$$

where $\mathbf{S} = \mathbf{D} - \mathbf{C}\mathbf{A}^{-1}\mathbf{C}^T$ is the Schur complement. Eq. 9 implies that since $\mathbf{A}^{-1}$ is known, we only need to inverse $\mathbf{S}$ to get the inverse of the left-hand side (with some extra matrix multiplications). As $\mathbf{S} \in \mathbb{R}^{M\times M}$, inversion of $\mathbf{S}$ thus takes negligible time, so the total time for evaluating Eq. 9 is at most $O(N^2)$ due to the extra multiplications.

With this proposed technique, the designer could interactively see the effect of adding each new constraint point on the fitted shape with much shorter delay time.

3 Approximating Interpolating Functions using Neural Net

The exact methods above still always require solving a linear system (i.e. inverting a matrix). Using statistical approximations, we do not need to solve linear systems at all. When for example the interpolating function f is $\mathbb{R}^2 \to \mathbb{R}$, let us first represent f as a neural net function $z = f_{\boldsymbol{\theta}}(x, y)$, where $\boldsymbol{\theta}$ is the trainable weights of the neural net.

We then define the loss function as:

$$E(f) = \sum_{i=1}^{N} \|f(x_i, y_i) - z_i\|^2 + \lambda \int_{\Omega} f_{xx}^2(x,y) + 2f_{xy}^2(x,y) + f_{yy}^2(x,y)\ dxdy, \tag{10}$$

where the first term penalizes f for not exactly interpolating the constraint points, and the second term penalizes unsmooth f. λ is a tunable parameter for balancing between interpolating accuracy and smoothness. Ω is the region of interest, whose area can be assuume to be 1 without loss of generality. We approximate the second term by uniformly sampling P auxiliary points in Ω:

$$E(f) \approx \sum_{i=1}^{N} \|f(x_i, y_i) - z_i\|^2 + \frac{\lambda}{P} \sum_{j=N+1}^{N+P} f_{xx}^2(x_j, y_j) + 2f_{xy}^2(x_j, y_j) + f_{yy}^2(x_j, y_j)\ dxdy. \tag{11}$$

In other words, the N constraint points are only used to evaluate the first term, and the P auxiliary points are only used for the second term.

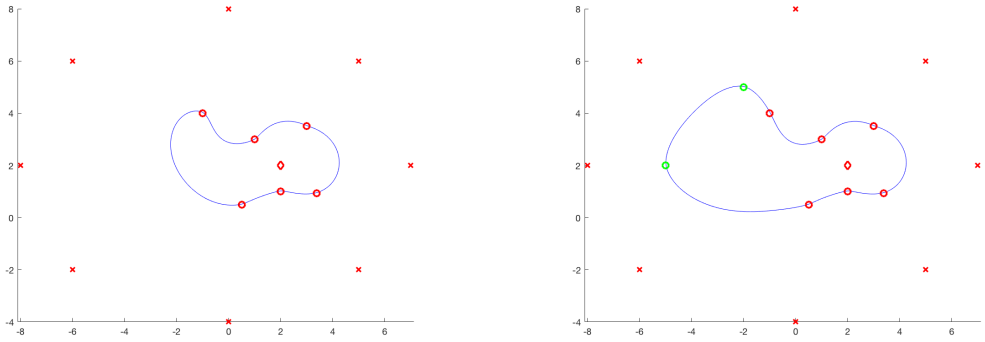
Gradient based methods are used to optimize $\boldsymbol{\theta}$ along the direction $-\nabla_{\boldsymbol{\theta}} E$. During training the target values (z) of the auxiliary points are "don't care"s. We shuffle the constraint points and auxiliary points so that each mini-batch more stably reflects gradient information from both terms. Though a wide-enough neural net is expressive to approximate any function, there is no guarantee with gradient methods that our final $f_{\boldsymbol{\theta}}$ will converge to the optimal solution. However, with standard mini-batch training, we are able to limit the total fitting time to $O(N + P)$.

4 Evaluations

We preformed experiments on fitting closed curves in 2D space using the implicit formulation. Fitting closed surfaces in 3D should be a straightforward extension and is not included in the scope of this mini-project. Since we use implicit formulation in 2D, the interpolating function f is $\mathbb{R}^2 \to \mathbb{R}$, and the ϕ function used in exact methods should be $\|\mathbf{x}\|^2 \log \|\mathbf{x}\|$. We use $z = 0$ for on-curve constraint points, $z = 1$ for interior points and $z = -1$ for exterior points.

4.1 Adding New Constraints using Block-inversion

After solving for the curve in Fig. 2 Left, we added two extra constraint points (Fig. 2 Right, Green) and applied the block-inversion technique to solve for the modified curve. We are able to exactly solve the modified curve without inverting the whole new matrix, as predicted by the derivations in Sec. 2.



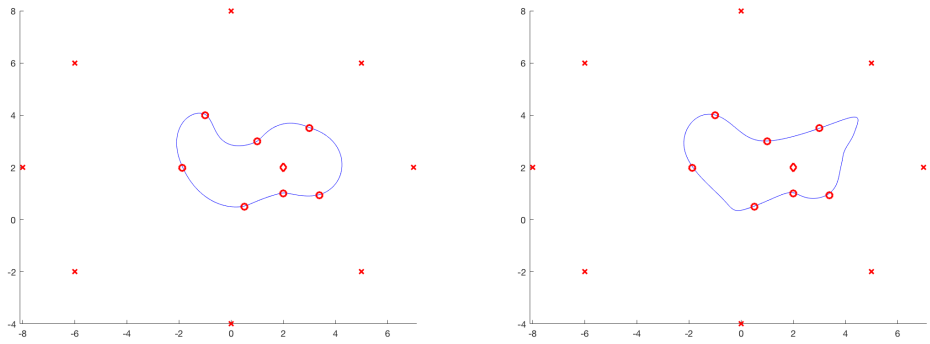
(a) Original curve solved from the constraint points. (b) New curve after adding 2 constraints (Green).

Figure 2: Adding new constraints using block-inversion. Same as before, diamonds represent inside constraints, crosses represent outside constraints, and circles represent on-curve constraints.

4.2 Using Neural Net Functions for Interpolation

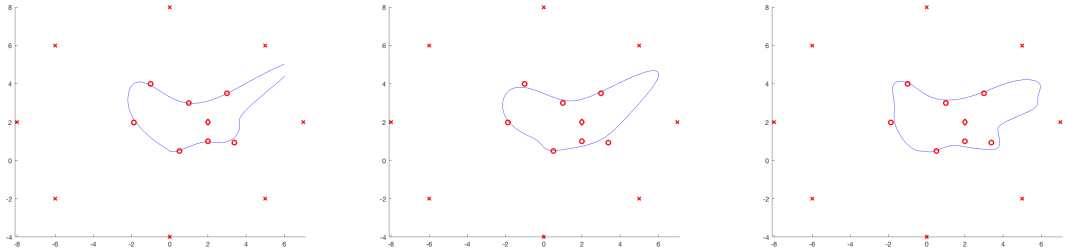
We compared the curves generated by our neural net approximators and by the classic exact method. We used a neural network with two hidden layers of 32 neurons each; input and output sizes are 2 and 1 respectively. $P = 250$ and $\lambda = 0.05$ in this experiment. As shown in Fig. 3, our approximated curve (Right) satisfies the constraint points almost exactly. Though two bumps exist at its southwest and northeast corners, the overall shape appears smooth and is closed. Note that constraint points in this example is quite sparse – none is placed on the right boundary.

Three baseline results of using neural nets differently are also presented in Fig. 4. For baseline (a), we set λ to zero in the loss function Eq. 10. As such, we only asked the neural net to perform a traditional regression on the constraint points, without requiring the regressed function to be smooth. For baseline (b), instead of evaluating the curvature loss at the uniformly sampled auxiliary points, we evaluated it at the constraint points. So no auxiliary points are needed for this baseline. For baseline (c), instead of using our curvature loss, we applied Dropout [4], a typical technique to regularize the landscapes of regressed neural net functions. All benchmarks show a visibly larger curvature than our approach, demonstrating the advantage of adding curvature loss and sampling auxiliary points.



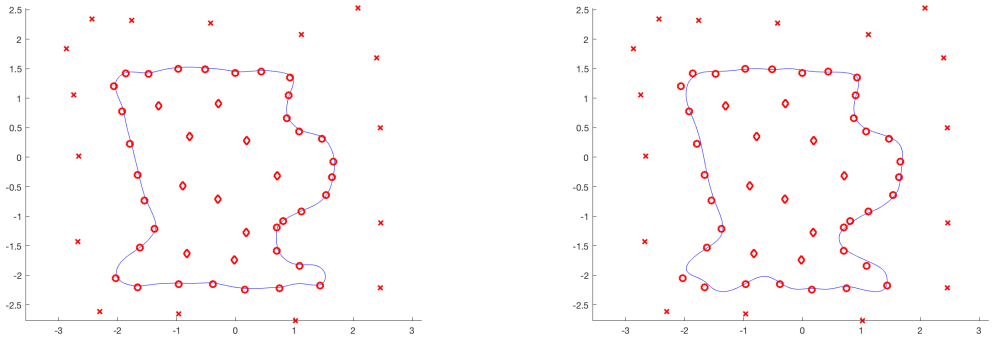
(a) Curve solved with exact method. (b) Curve solved with neural net approximation.

Figure 3: Using neural net function for interpolation.



(a) No curvature loss. (b) No auxiliary points. (c) Dropout regularization.

Figure 4: Baseline results of using neural net approximators differently than our approach.



(a) Curve solved with exact method. (b) Curve solved with neural net approximation.

Figure 5: Using neural net function for reconstructing a cup shape.

We finally applied our approximation approach to a real-world medium-scale problem, where the goal is to reconstruct the shape of a 2D cup from an array of noisy constraint points. As shown in Fig. 5, our approach is able to obtain a similar result with the exact method.

5 Conclusion

In this mini-project, we explored two methods for accelerating the calculation of interpolating implicit curves. The first one extends the classic exact thin-plate method using block-inversion theorem, so that adding new constraints to an already solved curve would not result in solving the whole linear system again. The second method uses a neural net to approximate the interpolation function, with a loss function designed to balance between interpolating accuracy and function smoothness.

References

[1] M. Powell, "Some algorithms for thin plate spline interpolation to functions of two variables," in *Advances In Computational Mathematics: New Delhi, India-Proceedings Of The Conference*, vol. 4, p. 303, World Scientific, 1994.

[2] J. Duchon, "Splines minimizing rotation-invariant semi-norms in sobolev spaces," in *Constructive theory of functions of several variables*, pp. 85–100, Springer, 1977.

[3] G. Turk and J. F. O'Brien, "Modelling with implicit surfaces that interpolate," *ACM Transactions on Graphics (TOG)*, vol. 21, no. 4, pp. 855–873, 2002.

[4] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, "Dropout: a simple way to prevent neural networks from overfitting," *The Journal of Machine Learning Research*, vol. 15, no. 1, pp. 1929–1958, 2014.