Machine Learning

Neural Networks

S. Nõmm

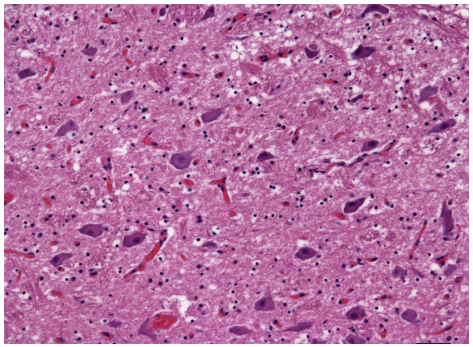
[1]Department of Software Science, Tallinn University of Technology

03.04.2018

What is *Artificial Neural Network ?*

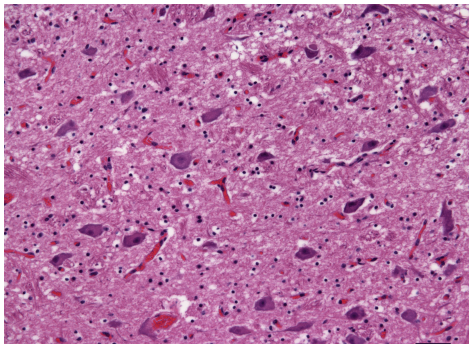
Biology inspired mathematical abstraction

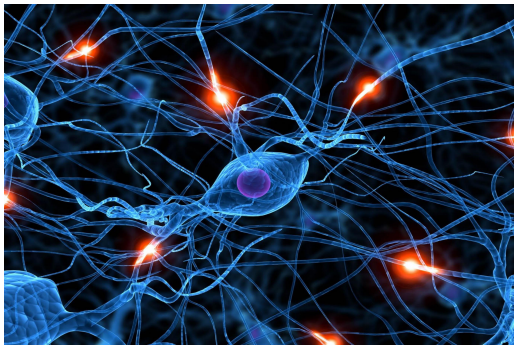
What is *Artificial Neural Network ?*



Biology inspired mathematical abstraction

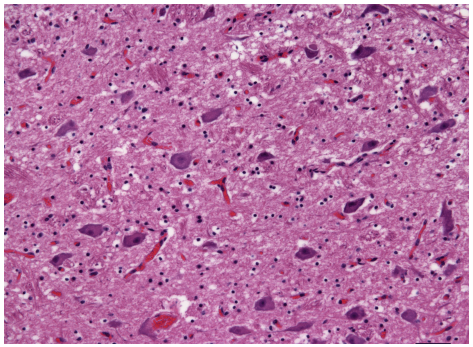
What is *Artificial Neural Network ?*

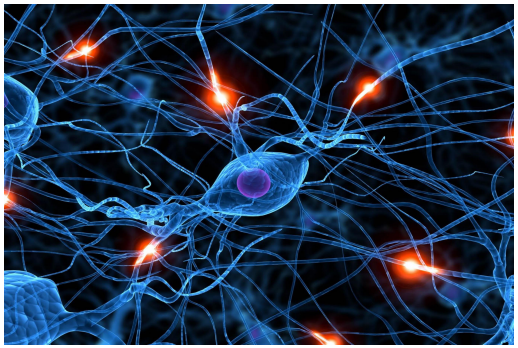




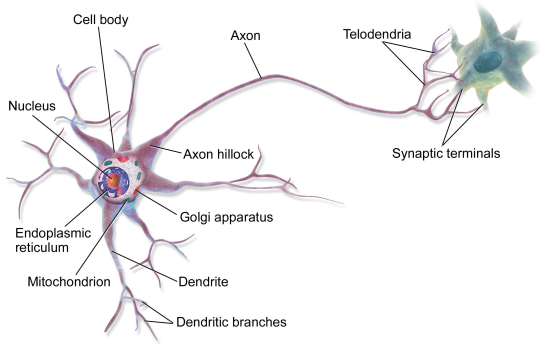
Biology inspired mathematical abstraction

What is *Artificial Neural Network ?*

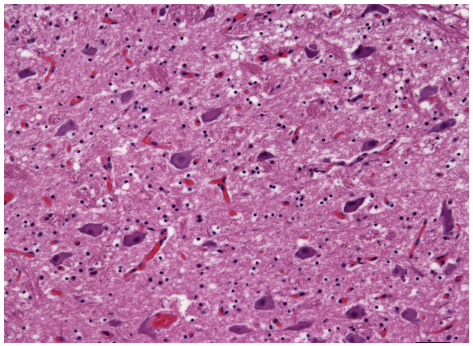


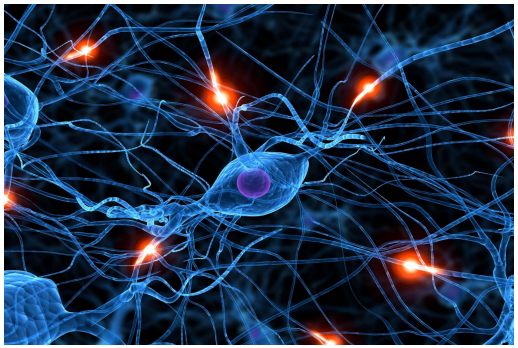


Biology inspired mathematical abstraction

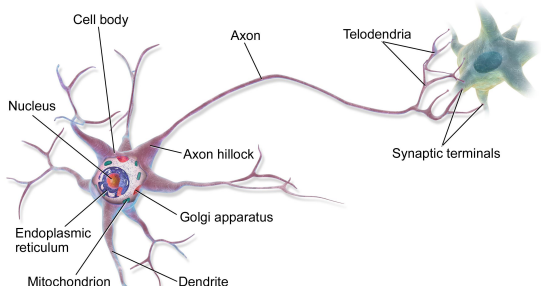


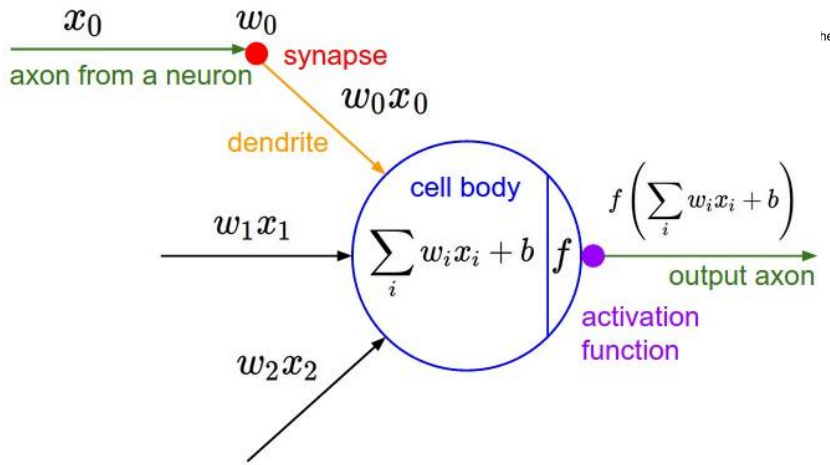
What is *Artificial Neural Network ?*



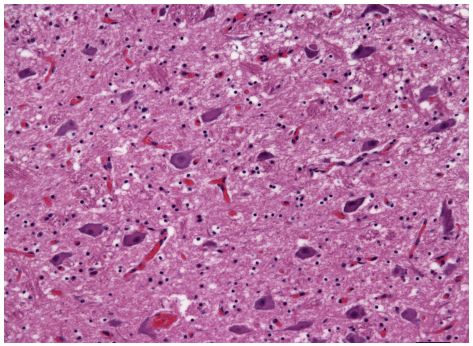


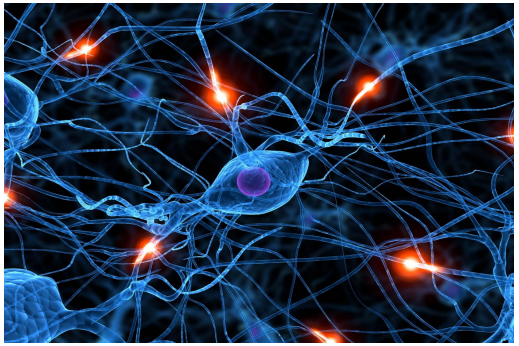
Biology inspired mathematical abstraction



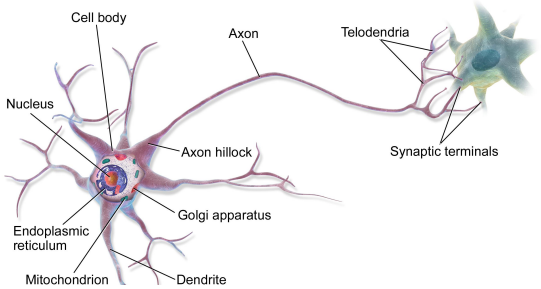


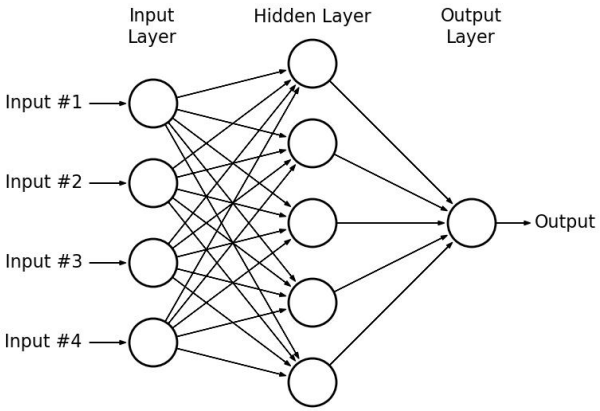
What is *Artificial Neural Network ?*

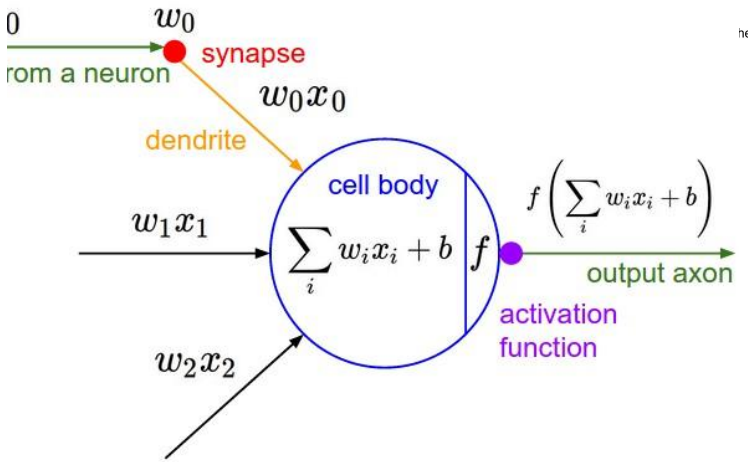




Biology inspired mathematical abstraction







Artificial Neural Network

- Brain consists of neurons.
- Basic idea of the artificial neural network is to combine models of the single neurons.
- Mathematical model of the single neuron is called *perceptron*.
- Perceptron has a number of drawbacks motivating the idea of artificial neural networks.

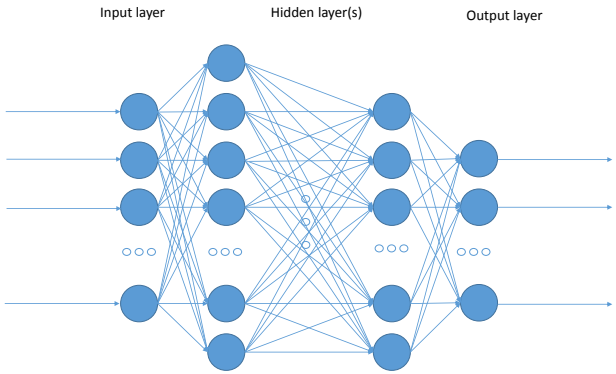
The model of a single neuron

$$y = f\left(\sum_{j=1}^{d} w_j x_j\right) = f\left(w^T x\right)$$

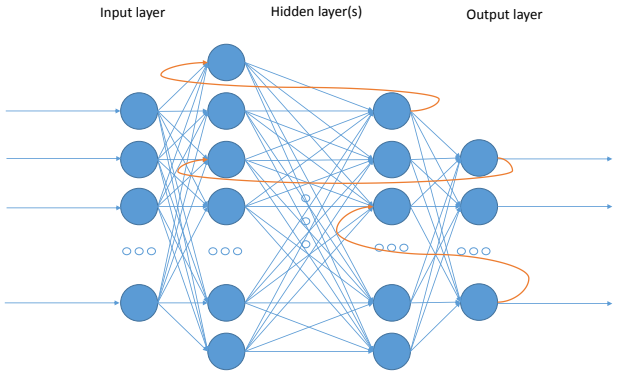
- d is the dimension of the input vector and number of *weights* w_j.
- $x \in \mathbb{R}^d$ - *input vector*.
- $f(\cdot)$ is a smooth nonlinear function referred as *activation function*. Most popular activation functions are:
  - Log-Sigmoid function $logsig(x) = 1/(1 + e^{-x})$.
  - Tan-Sigmoid function $tansig(x) = 2/(1 + e^{(-2*n)}) - 1$
  - Linear transfer function
- In case of Log -Sigmoid activation function the model of the neuron takes form:

$$y = \frac{1}{1 + e^{-\sum_{j=1}^{d} w_j x_j}}$$

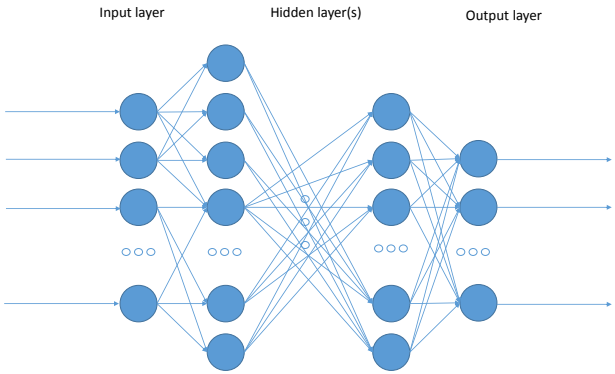
Different topologies of the ANN



Fully connected feed forward neural network



Recurrent neural network



Restricted connectivity feed forward neural network

ANN

- How to choose optimal topology of the net? Nr of layers? Nr of neurons on hidden layer etc?
- Universal approximation theorem:
  - G.Cybenko (1989) Single hidden layer with finite number of neurons (multilayer perceptron) can approximate continuous functions on a compact subsets of $\mathbb{R}^n$. Some weak assumptions about activation function were made. Algorithmic aspects were not touched.
  - K.Hornik (1991) Demonstrated importance of the choice of architecture over the choice of activation function.

Steps specific to the modelling of NN

- Initialization of the network.
- Training algorithms.
  - Levenberg-Marquardt backpropagation.
  - Bayesian regularization backpropagation.
  - Scaled conjugate gradient backpropagation.
  - Resilent backpropagation.
- Stopping criteria.
- Epoch - is the measure of the number of times all the training vectors are used to update the weights.

Training

- Training is iterative process which starts with initialization, when initial weights are generated randomly or defined using some other technique.
- During each iteration (epoch) measure of imprecision (loss function (usually nonlinear)) is calculated then values of the weights are updated. The idea is to solve optimization problem with respect to loss function.
- Iterations are repeated until stopping criteria is met.
- The choice of loss function, number of weights to identify and computational restrictions define the choice of the training algorithm.
- Gradient descent, Newton's method, Conjugate gradient, Quasi Newton method, Levenberg - Marquardt algorithm,

Training Algorithms

- Gradient descent (steepest descent). Slow but does not require a lot of memory. Denote $\nabla f(w_i) = g_i$ then weights update rule is

$$w_{i+1} = w_i - g_i \cdot \eta_i, \quad i = 0, 1, \ldots$$

where η_i is the learning rate.

- Levenberg - Marquardt algorithm. Fast but requires a lot of memory. Loss function:

$$f = \sum e_i^2$$

where e_i are the residuals for each point of the data set. Jacobian matrix of the loss function:

$$J_{i,j} = \frac{\partial e_i}{\partial w_j}$$

where i indexes data points and j - weights of the network. Weights update rule (here i is the epoch identifier):

$$w_{i+1} = w_i - (J_i^T J_i + \lambda_i I)^{-1}(2 J_i e_i), \quad i = 0, 1, \ldots$$

where λ plays the role of a learning rate.

Example

- Data set is represented by the surface in 3D set.
- Let us adopt LM algorithm to just single neuron and observe convergence process step by step.
- Initialize the process by randomly generating the weights.
- Denote the sum of squared residuals as SSR

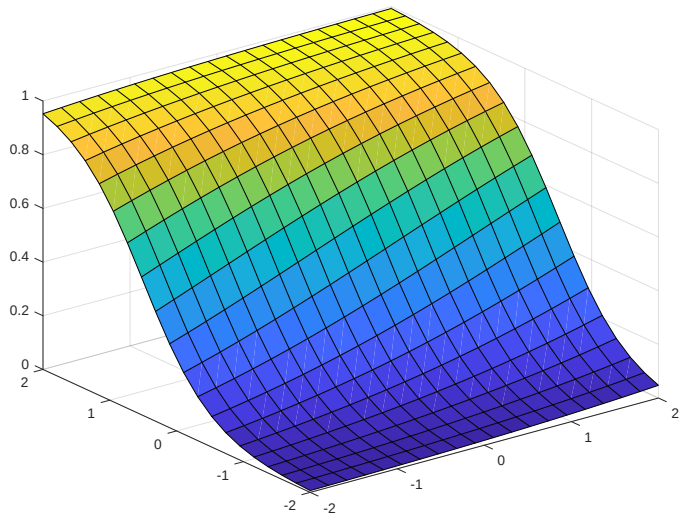
Neuron

Consider one neuron with two inputs and logsig activation function

- Consider one neuron with two inputs (denoted x and y) and logsig activation function. (Notation is chosen to simplify geometric interpretation).
- Mathematical model of such neuron is given by

$$z = \frac{1}{1 + e^{-(w_1 x + w_2 y)}}$$

- For the values $w_1 = 0.5$ and $w_2 = 2$ this function graph in 3D space is



Implementation

- Loss function:

$$f = \sum \left(z_i - \frac{e^{(w_1 x_i + w_2 y_i)}}{e^{(w_1 x_i + w_2 y_i)} + 1} \right)^2$$

- Jacobian matrix of the loss function:
  Elements of the first column:

$$j_{i,1} = \frac{\partial e_i}{\partial w_1} = 2 \left(\frac{-z_i x_i e^{(w_1 x_i + w_2 y_i)}}{(e^{(w_1 x_i + w_2 y_i)} + 1)^2} + \frac{x_i e^{2(w_1 x_i + w_2 y_i)}}{(e^{(w_1 x_i + w_2 y_i)} + 1)^3} \right)$$

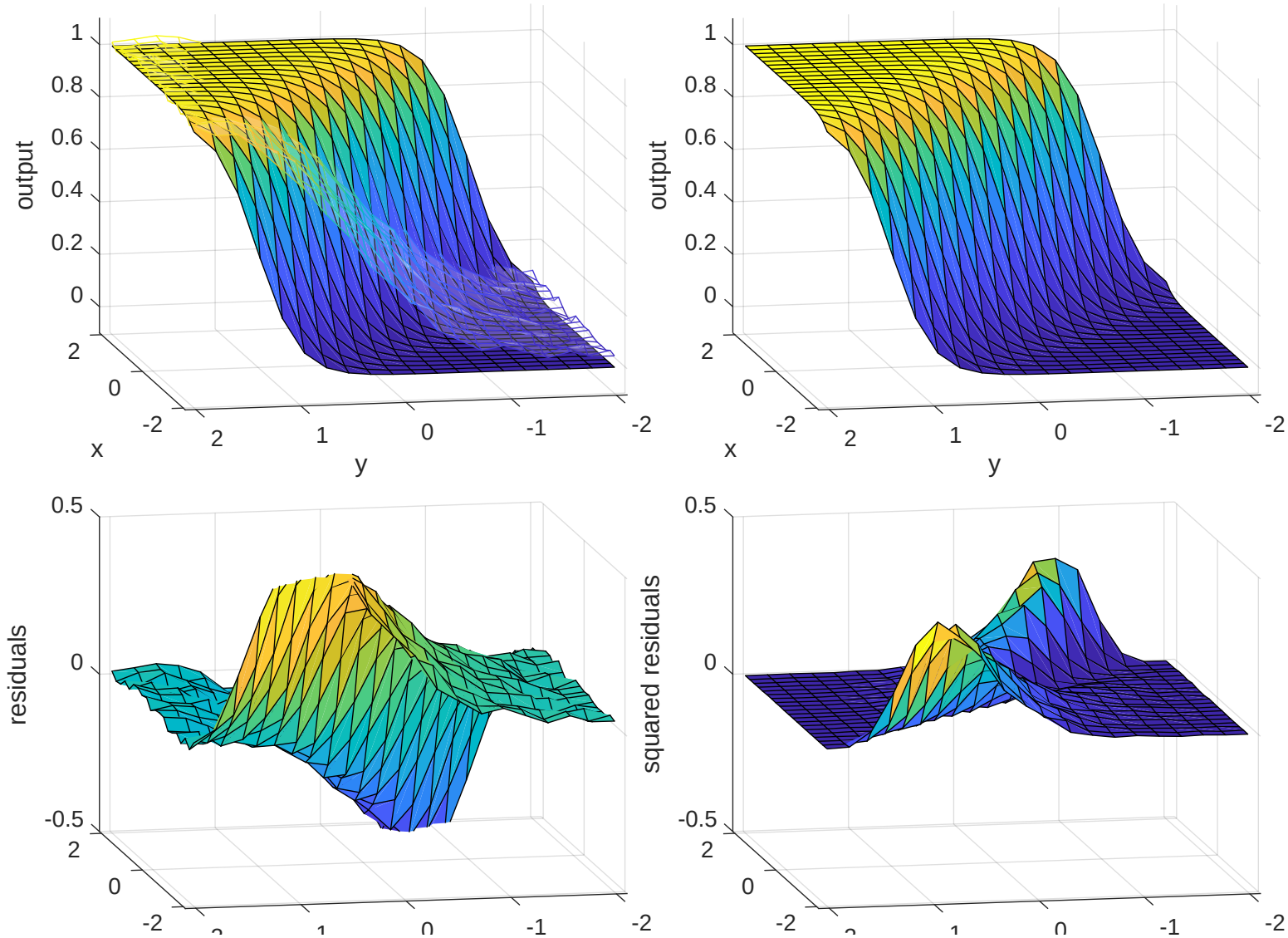
  Elements of the second column:

$$j_{i,2} = \frac{\partial e_i}{\partial w_2} = 2 \left(\frac{-z_i y_i e^{(w_1 x_i + w_2 y_i)}}{(e^{(w_1 x_i + w_2 y_i)} + 1)^2} + \frac{y_i e^{2(w_1 x_i + w_2 y_i)}}{(e^{(w_1 x_i + w_2 y_i)} + 1)^3} \right)$$

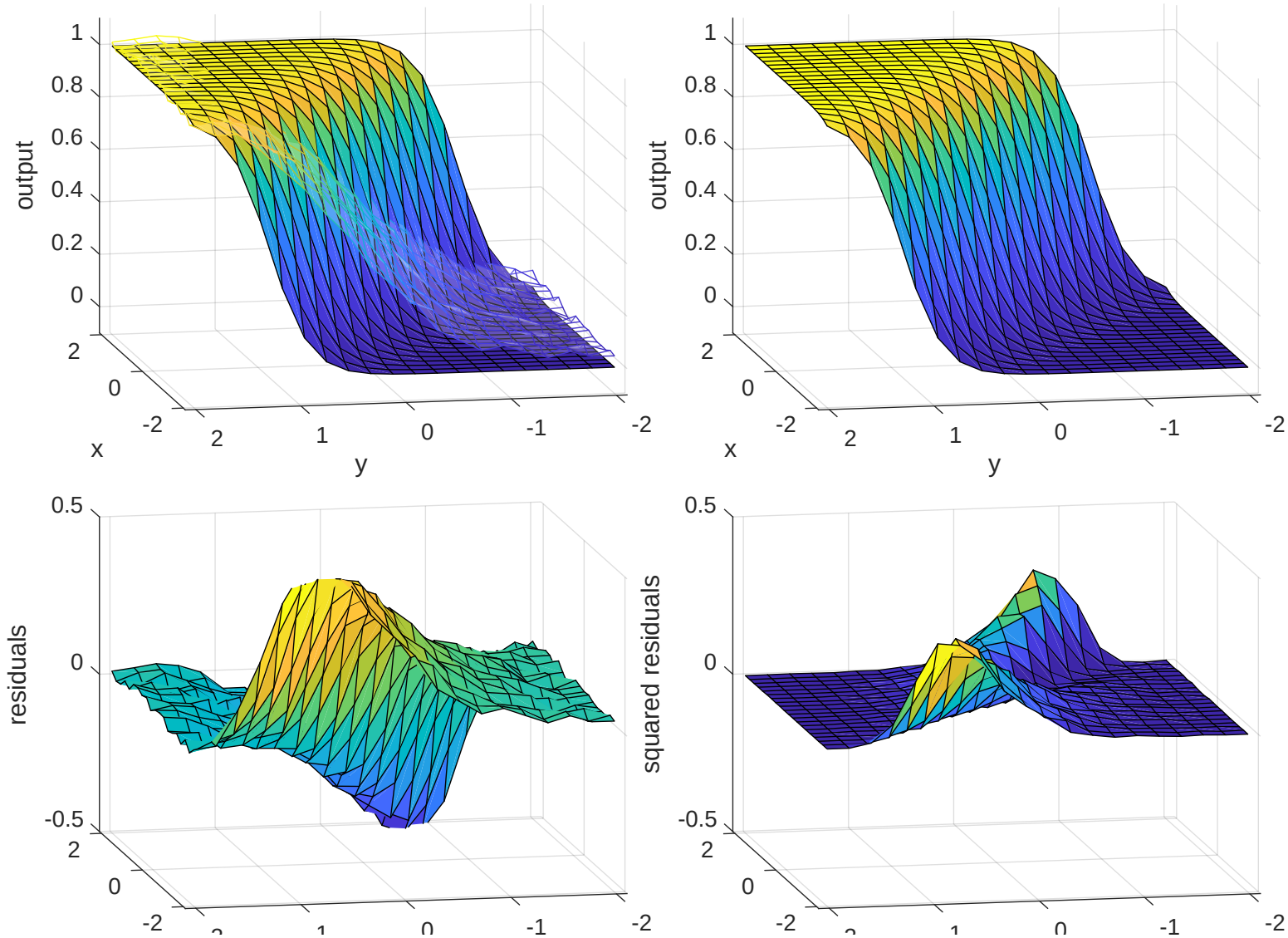
Neuron

- Generate the surface to approximate adding some noise.
- Levenberg - Marquardt algorithm uses sum of squared errors as the loss function.
- Initialize the process by randomly generating the weights.
- Denote sum of squared residuals as SSR

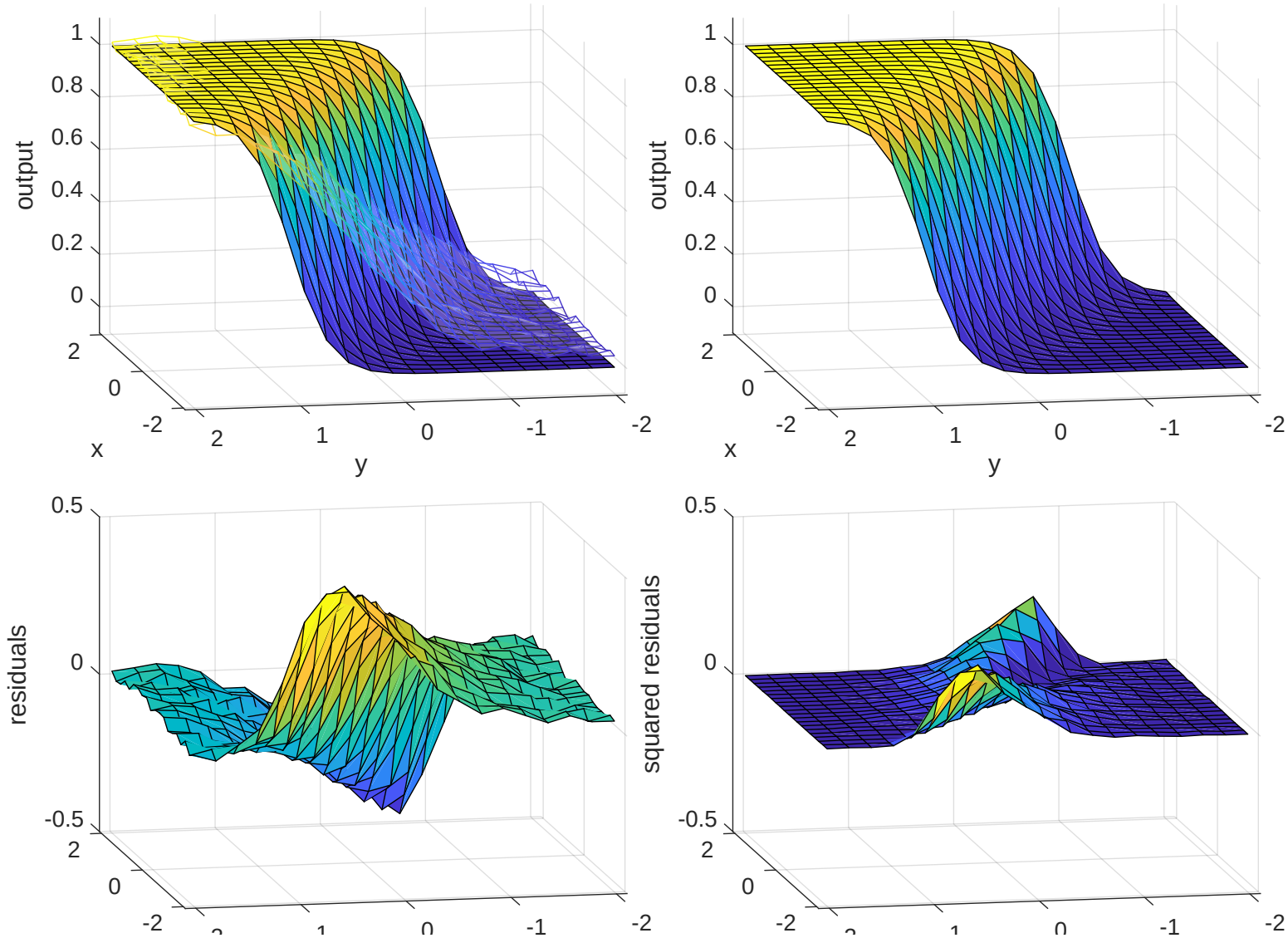
Initial guess, SSR $= 17.51$



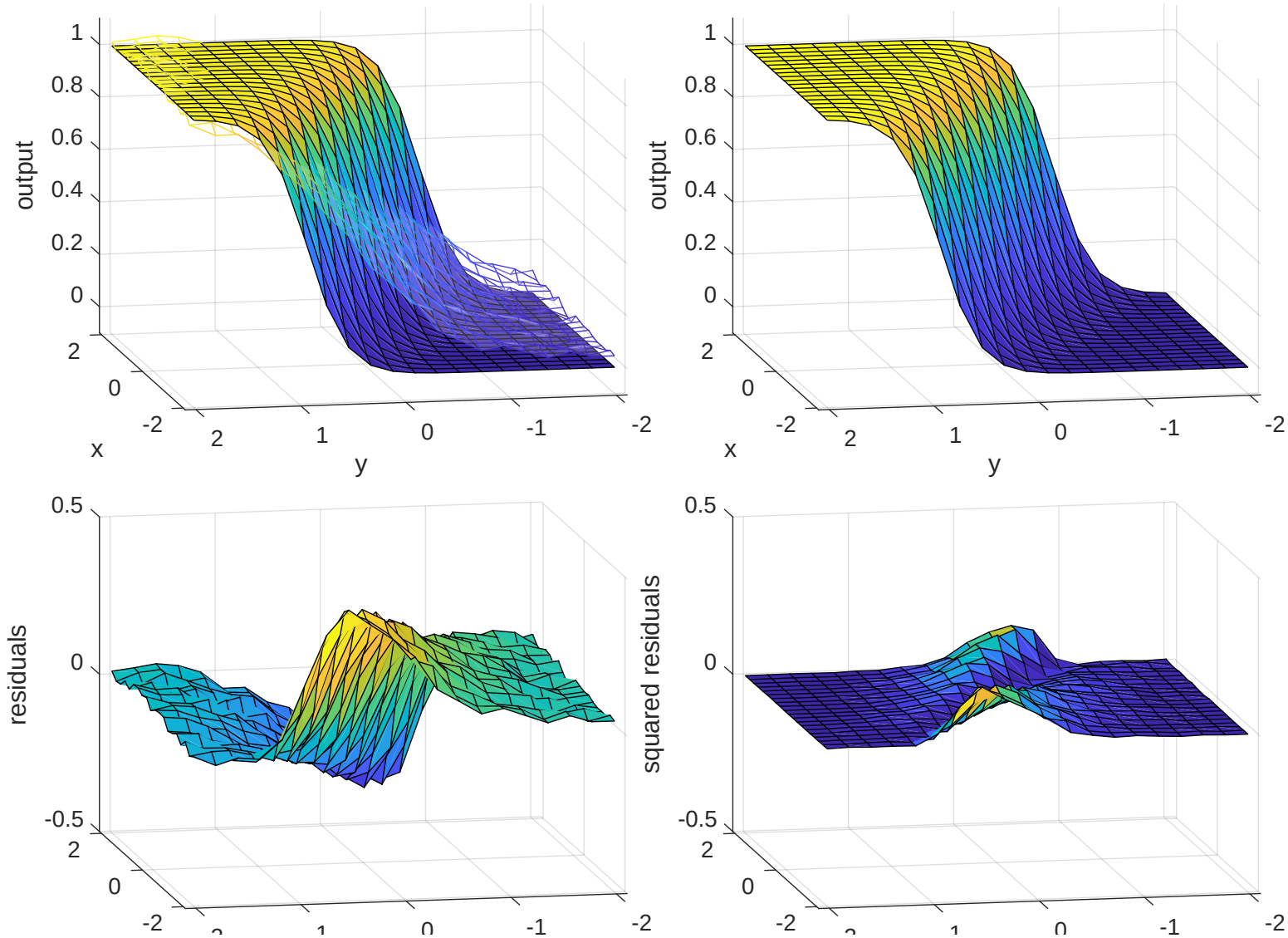
Epoch 2, SSR = 15.09



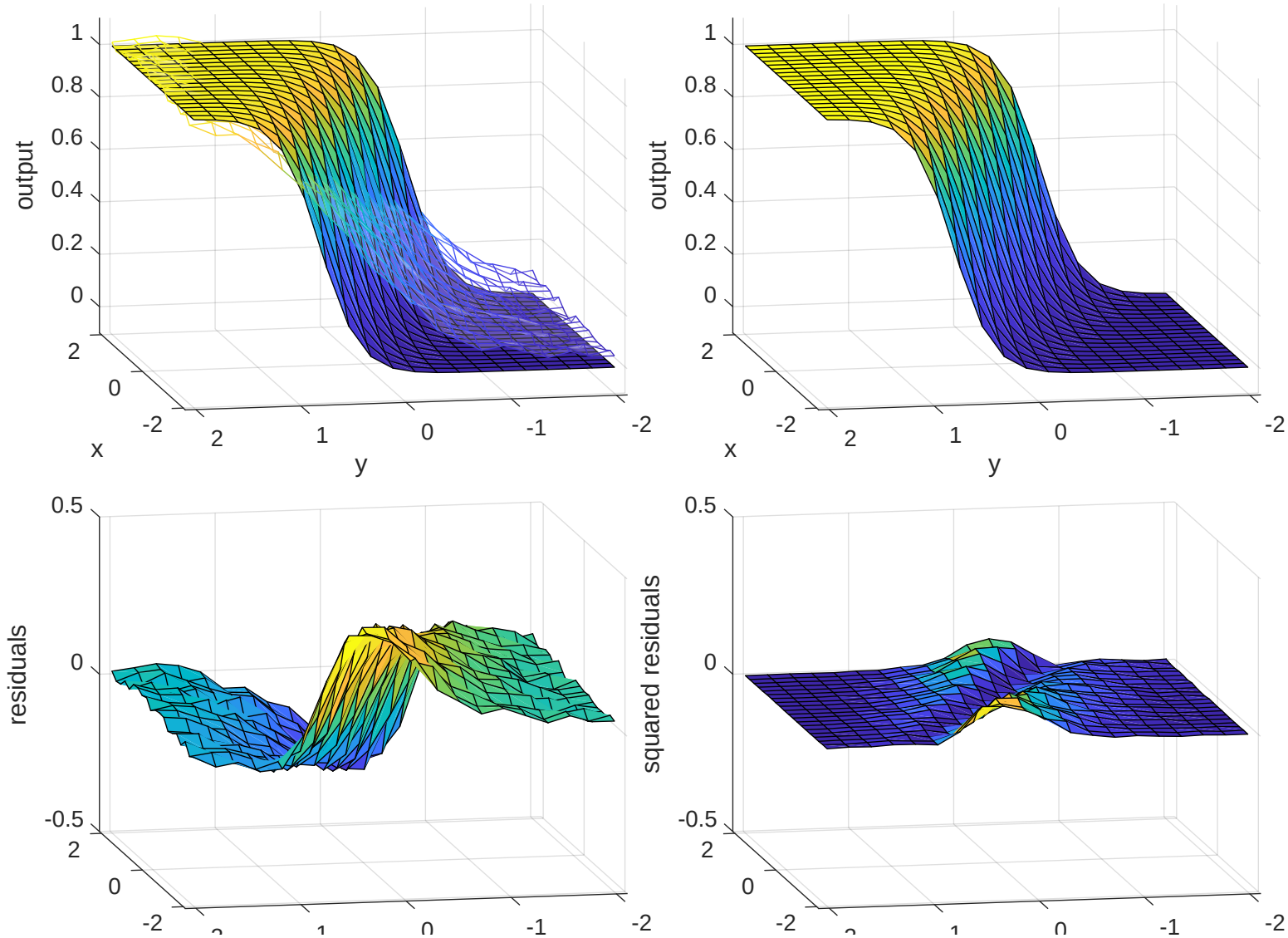
Epoch 4, SSR = 11.05



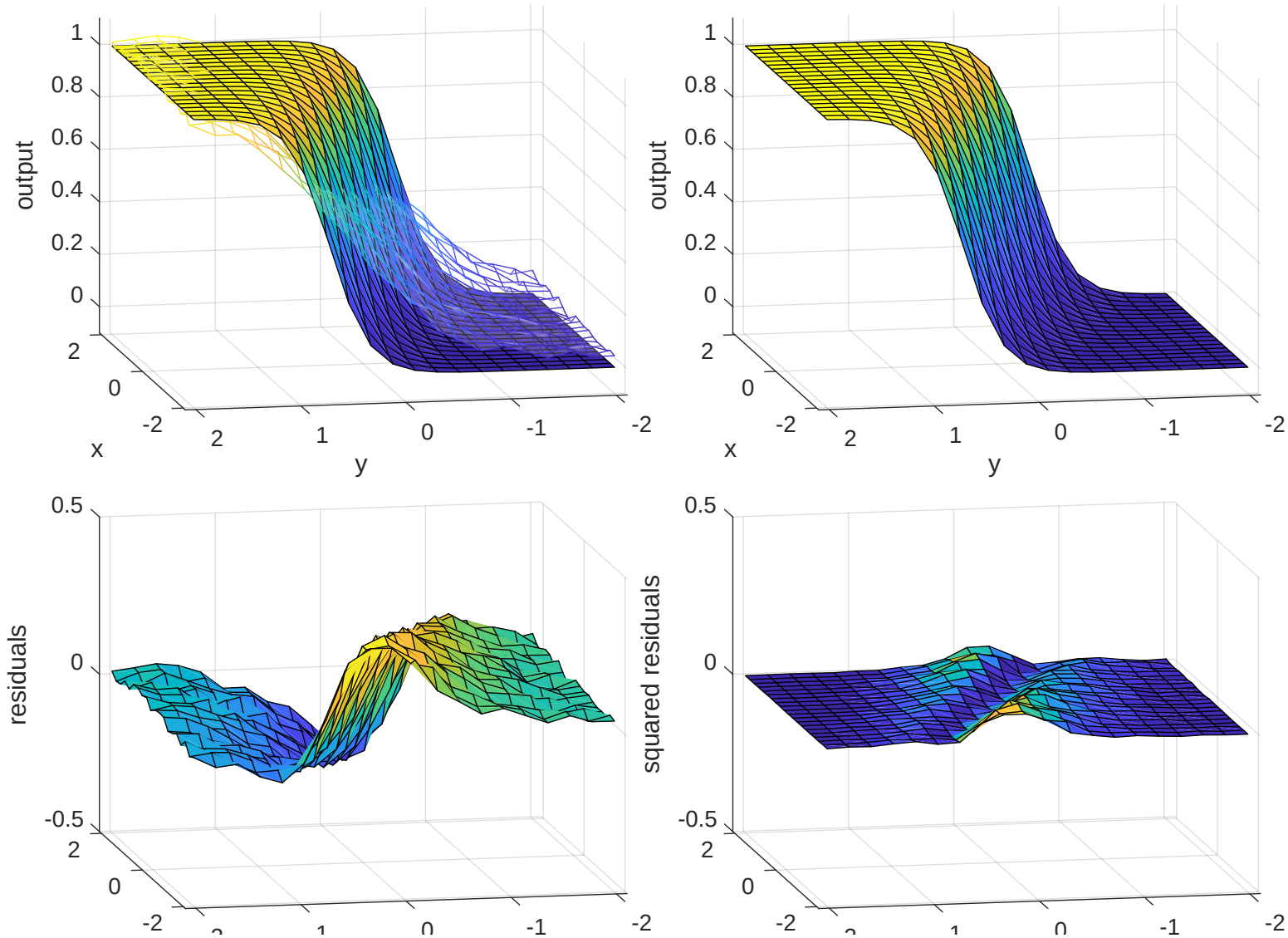
Epoch 6, SSR $= 8.45$



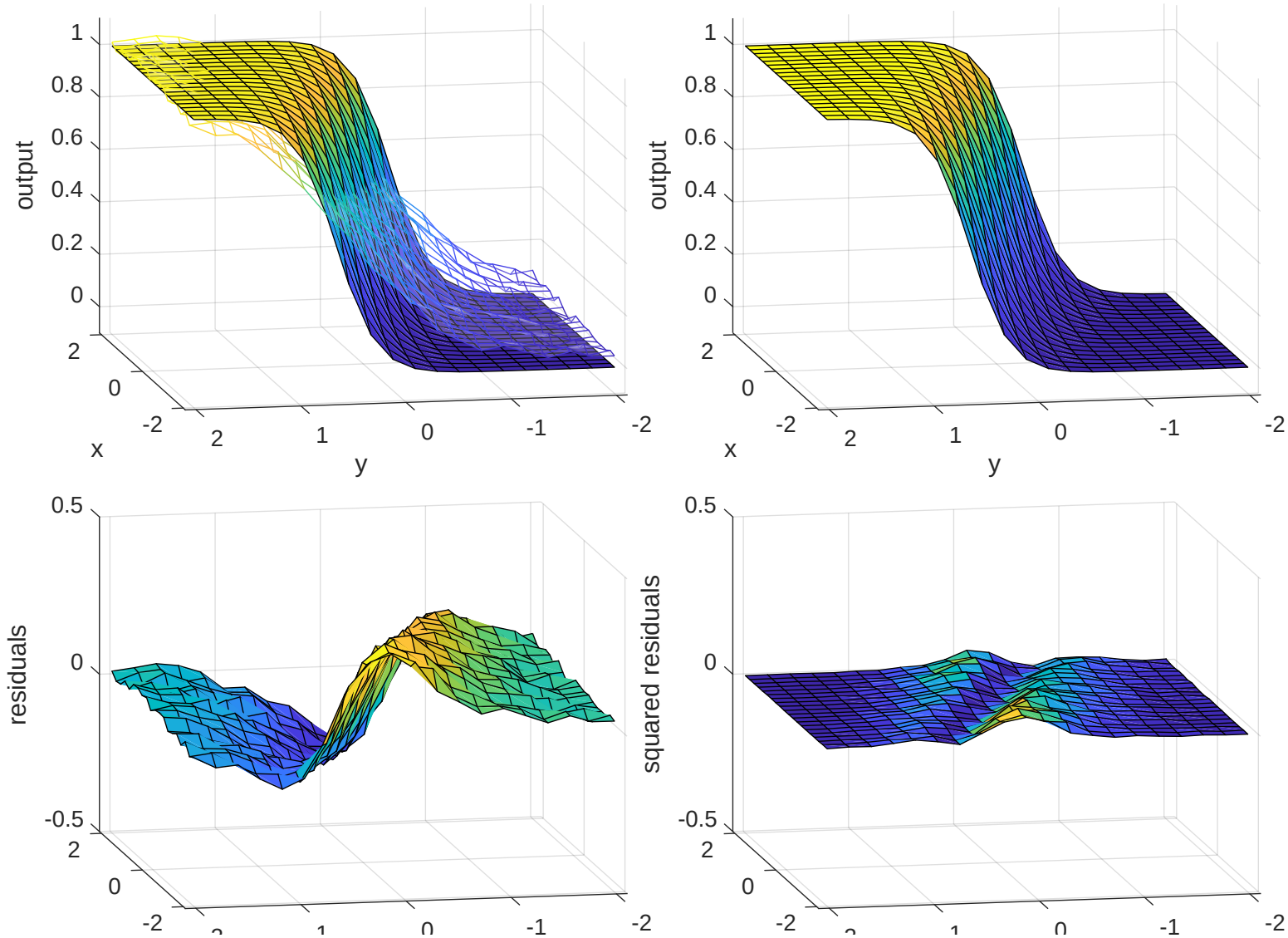
Epoch 8, SSR = 7.19



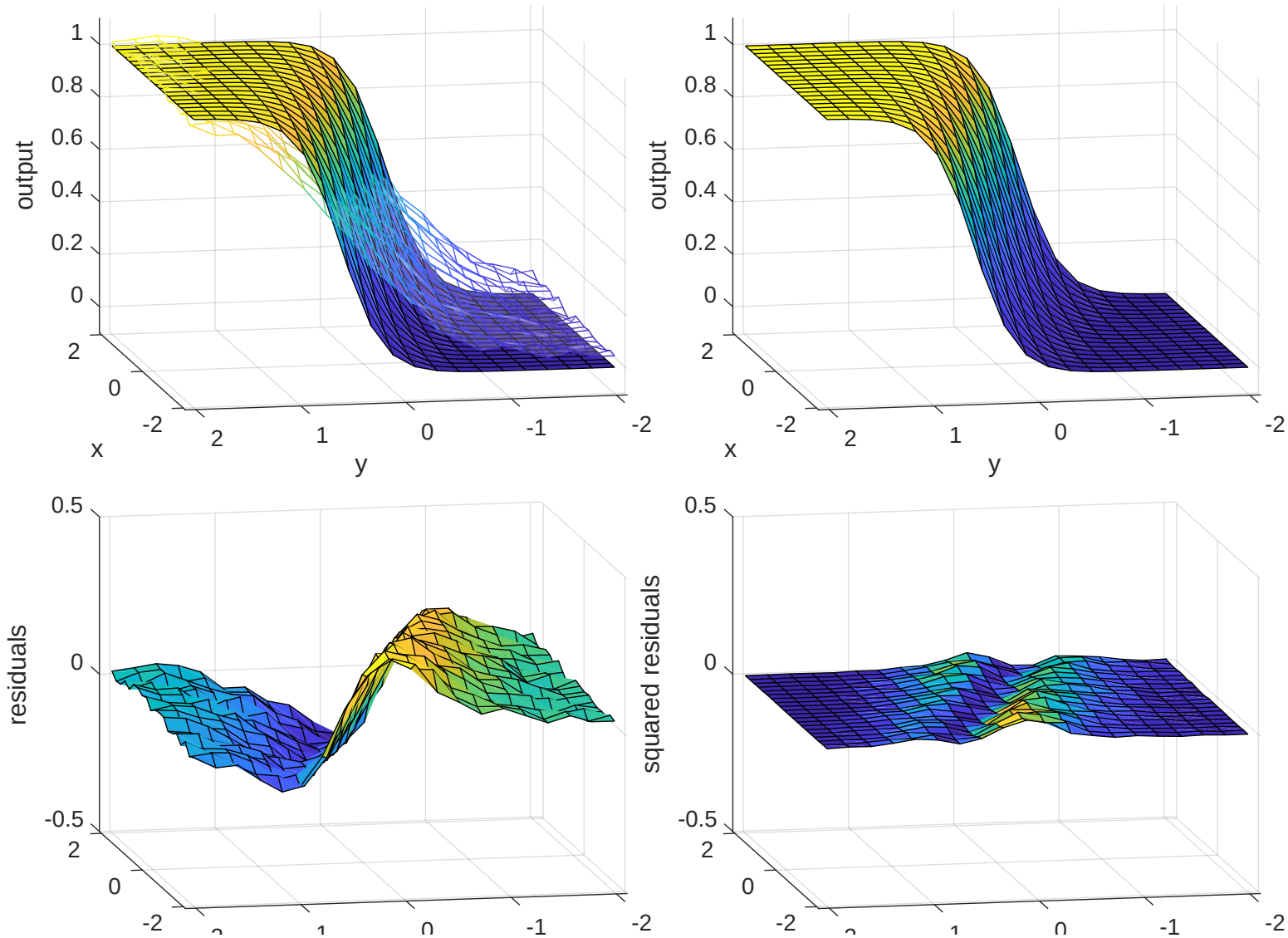
Epoch 10, SSR $= 6.55$



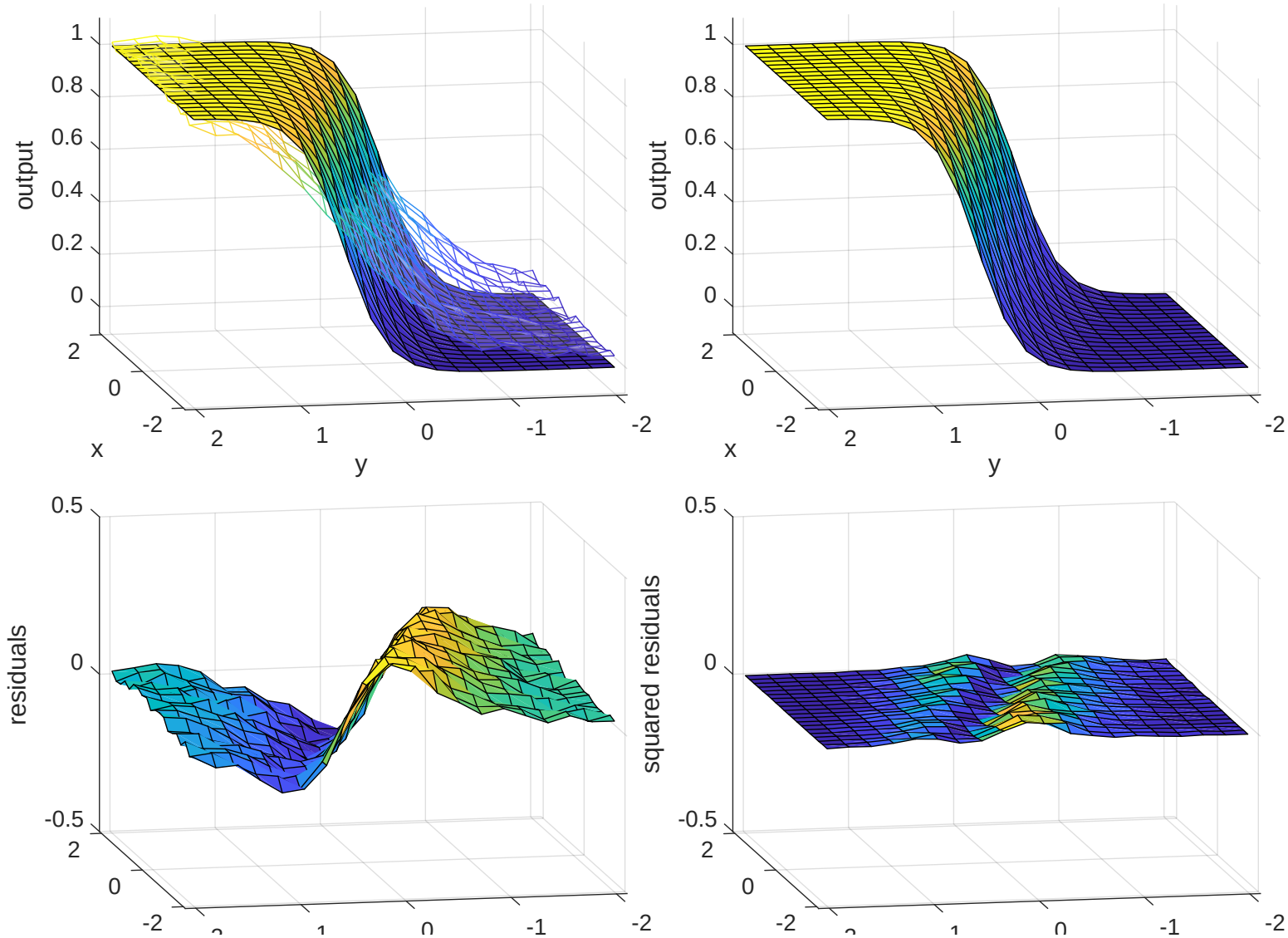
Epoch 12, SSR = 6.14



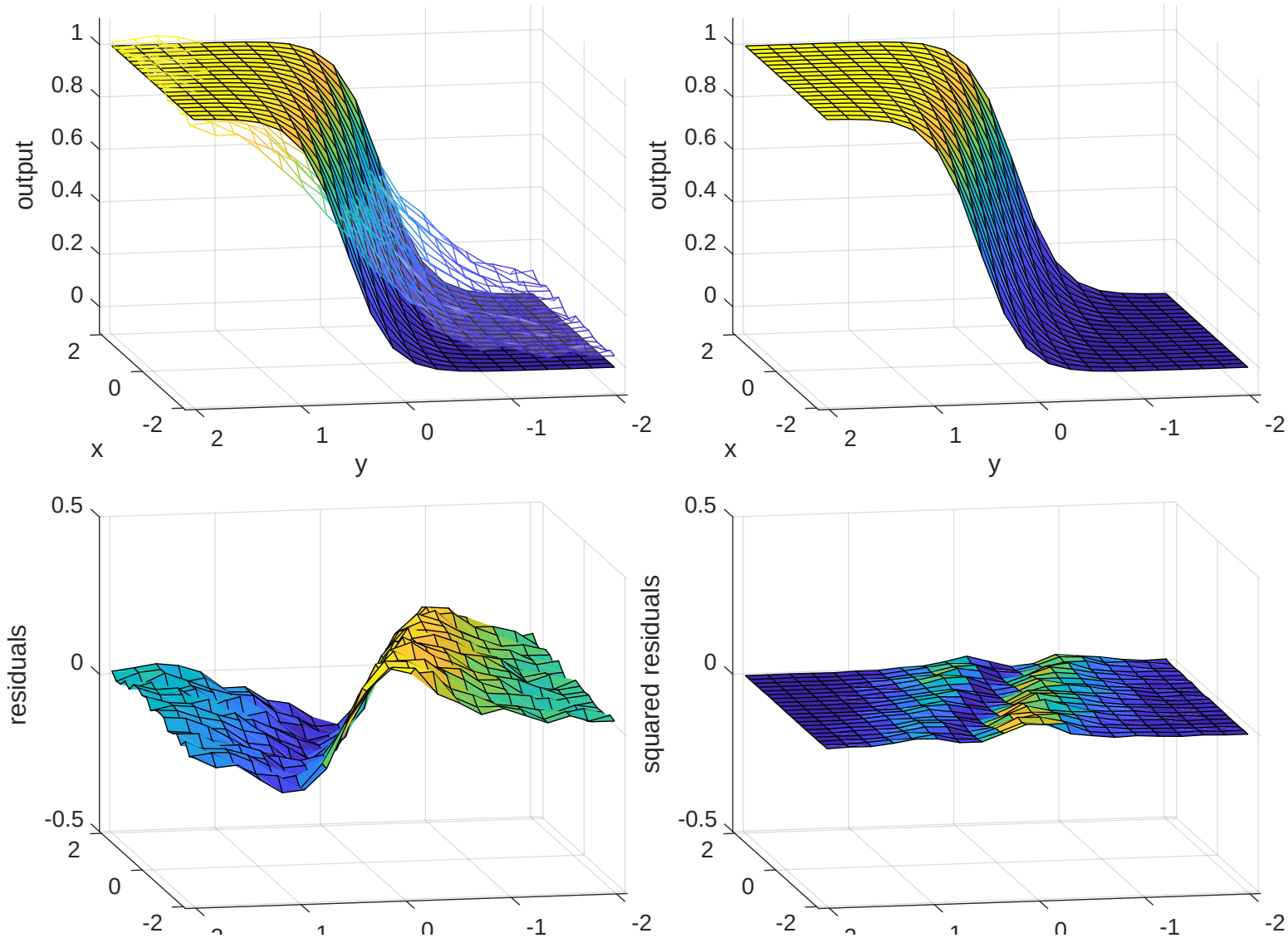
Epoch 14, SSR $= 5.82$



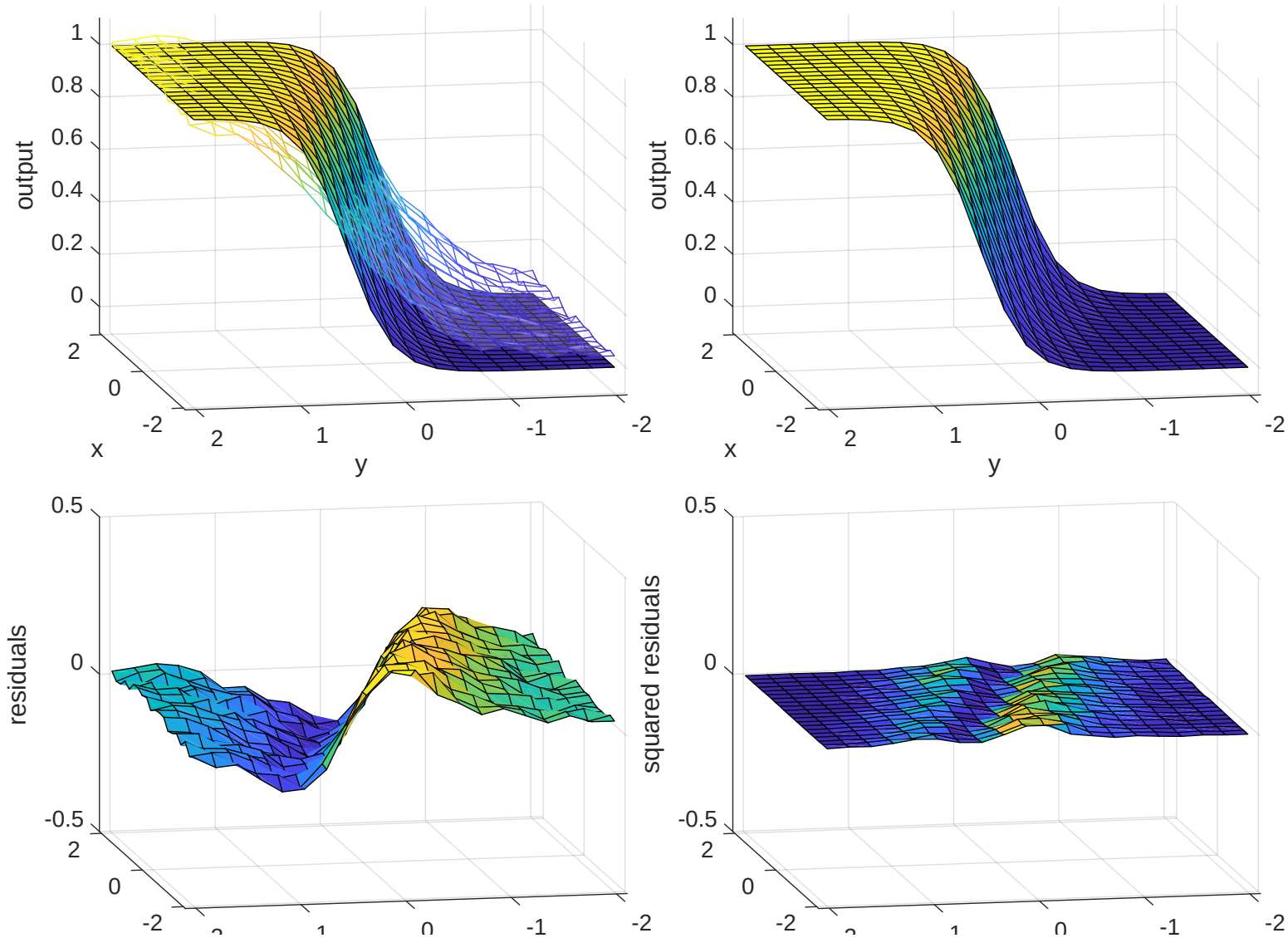
Epoch 16, SSR $= 5.53$



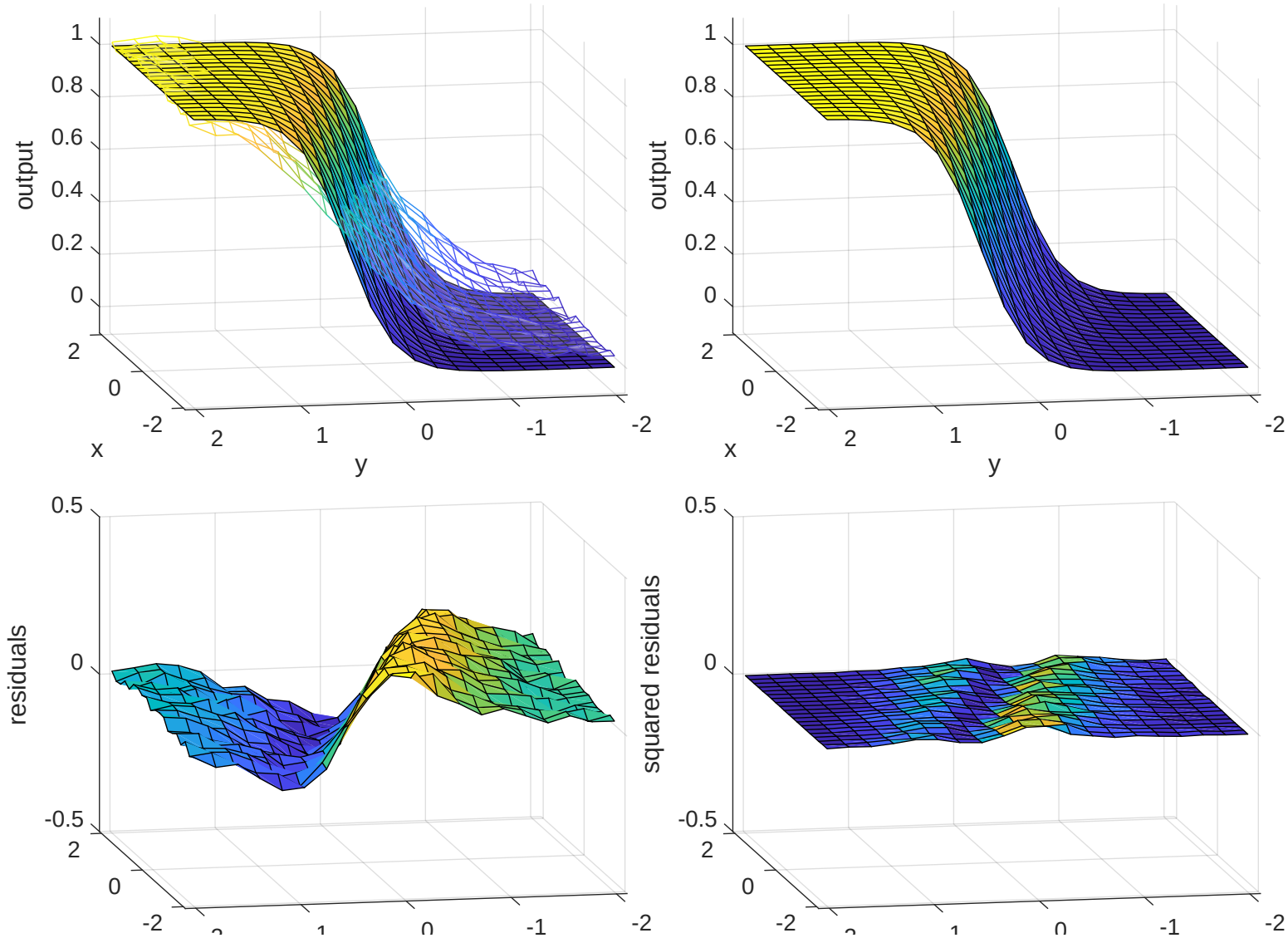
Epoch 18, SSR $= 5.26$



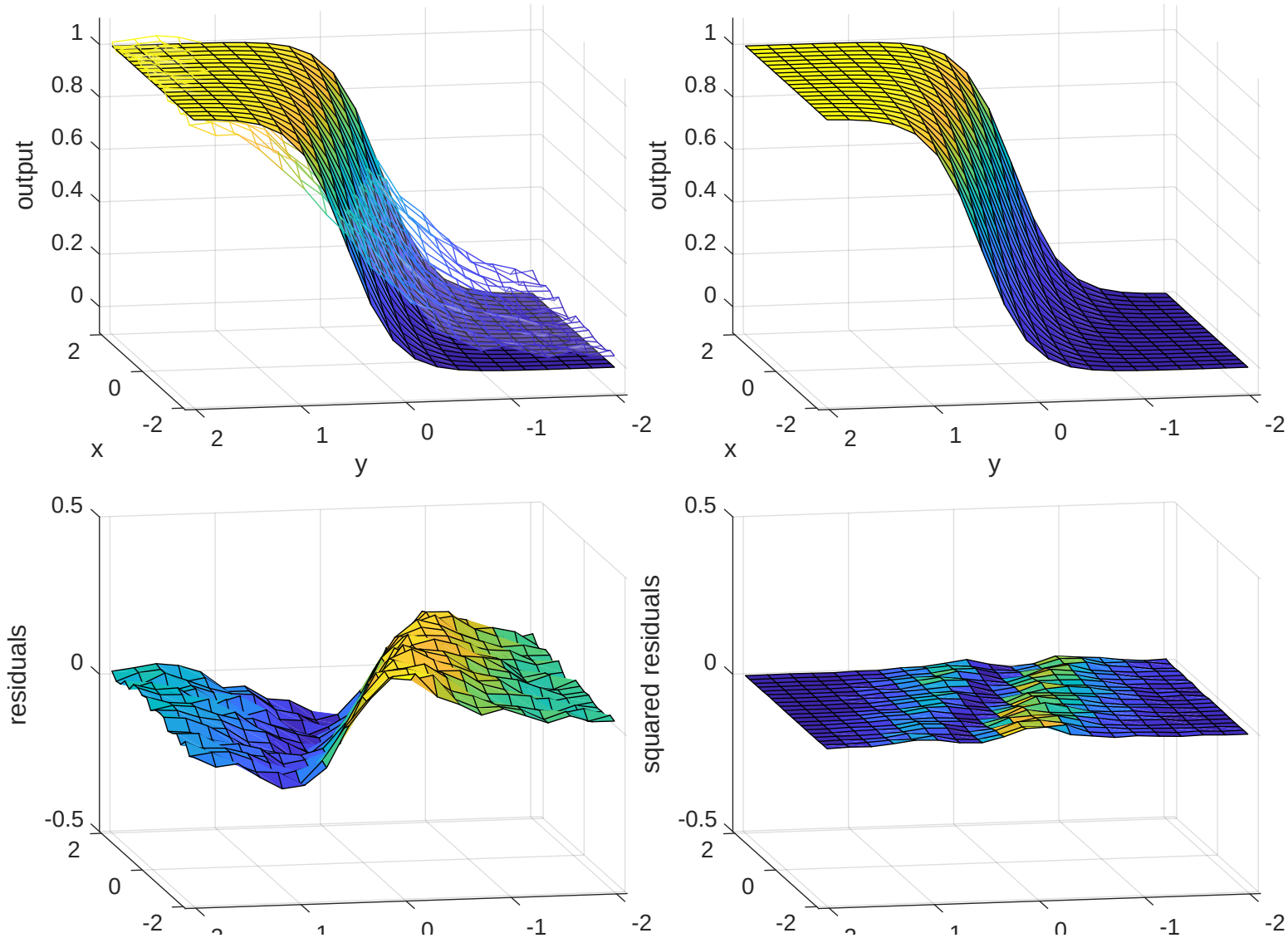
Epoch 20, SSR $= 4.98$



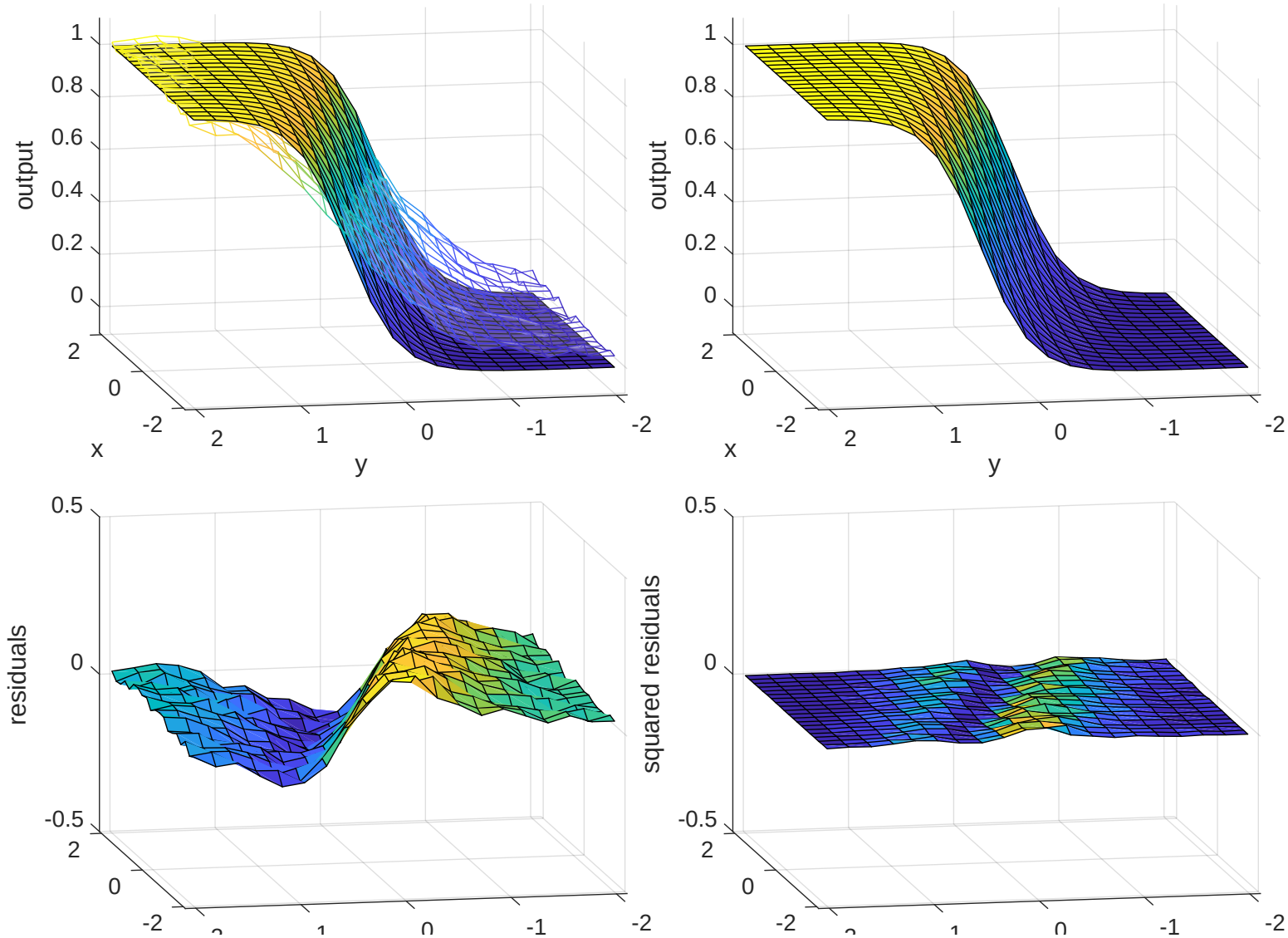
Epoch 22, SSR $= 4.69$



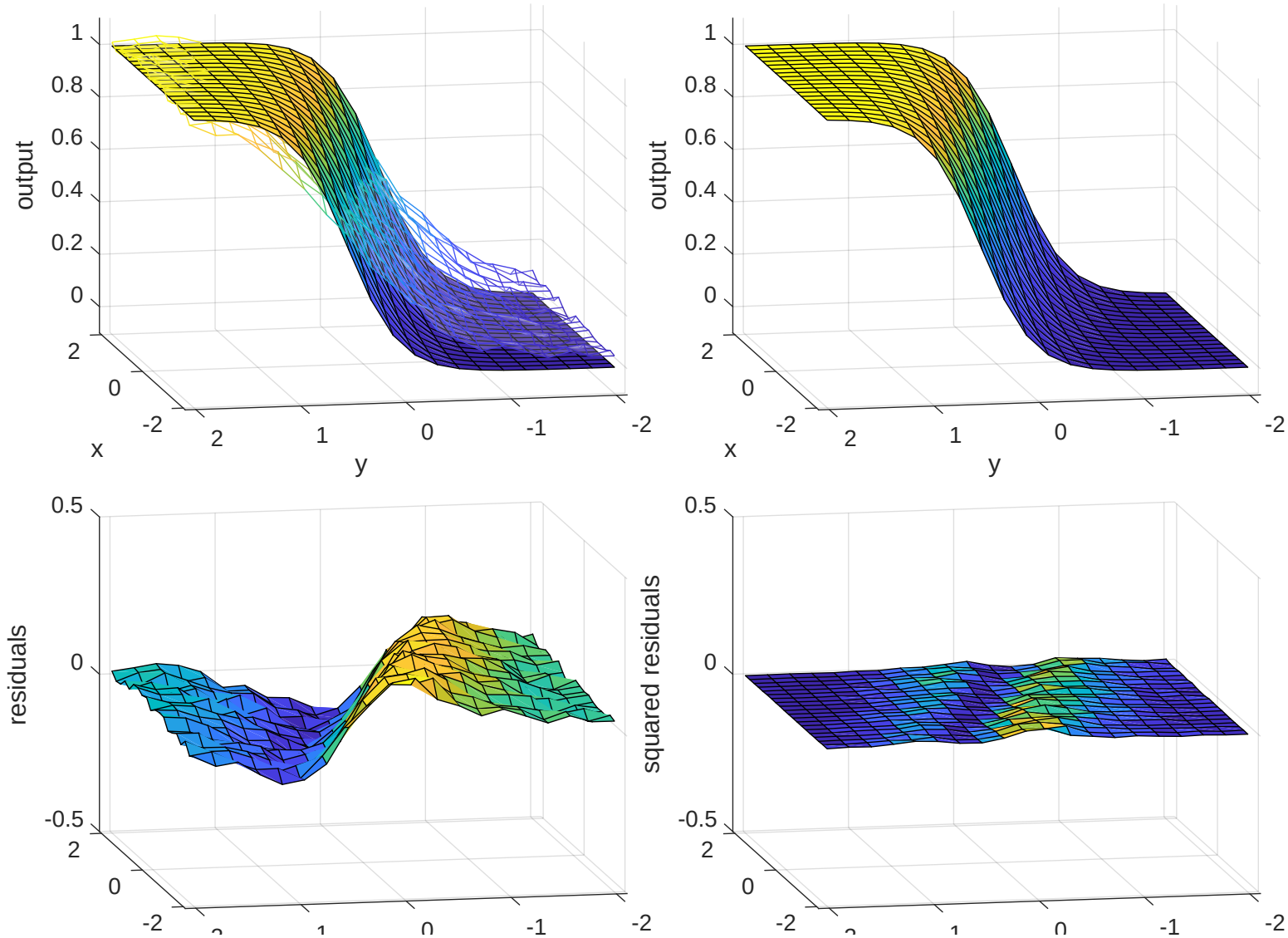
Epoch 24, SSR $= 4.4$



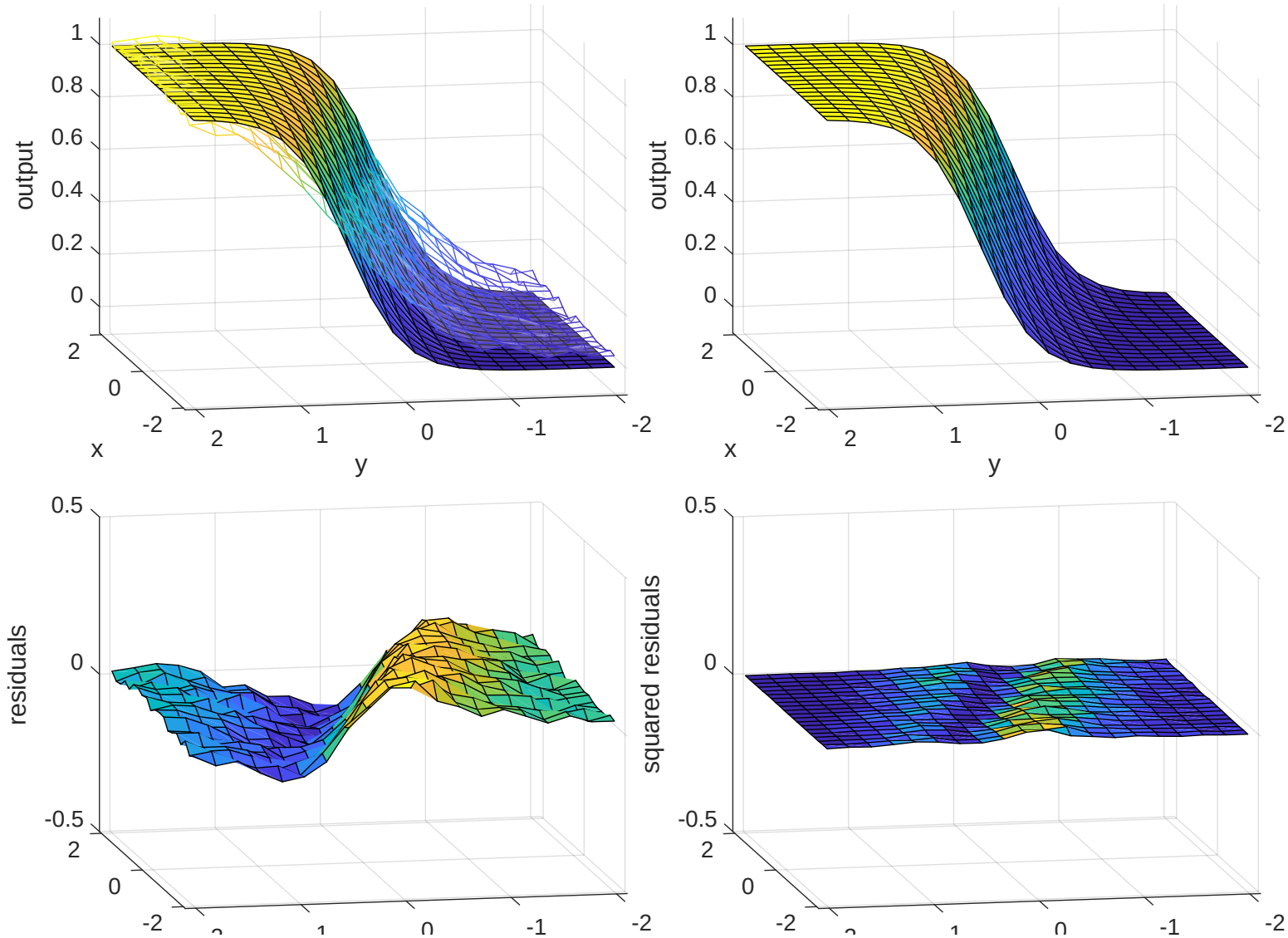
Epoch 26, SSR $= 4.08$



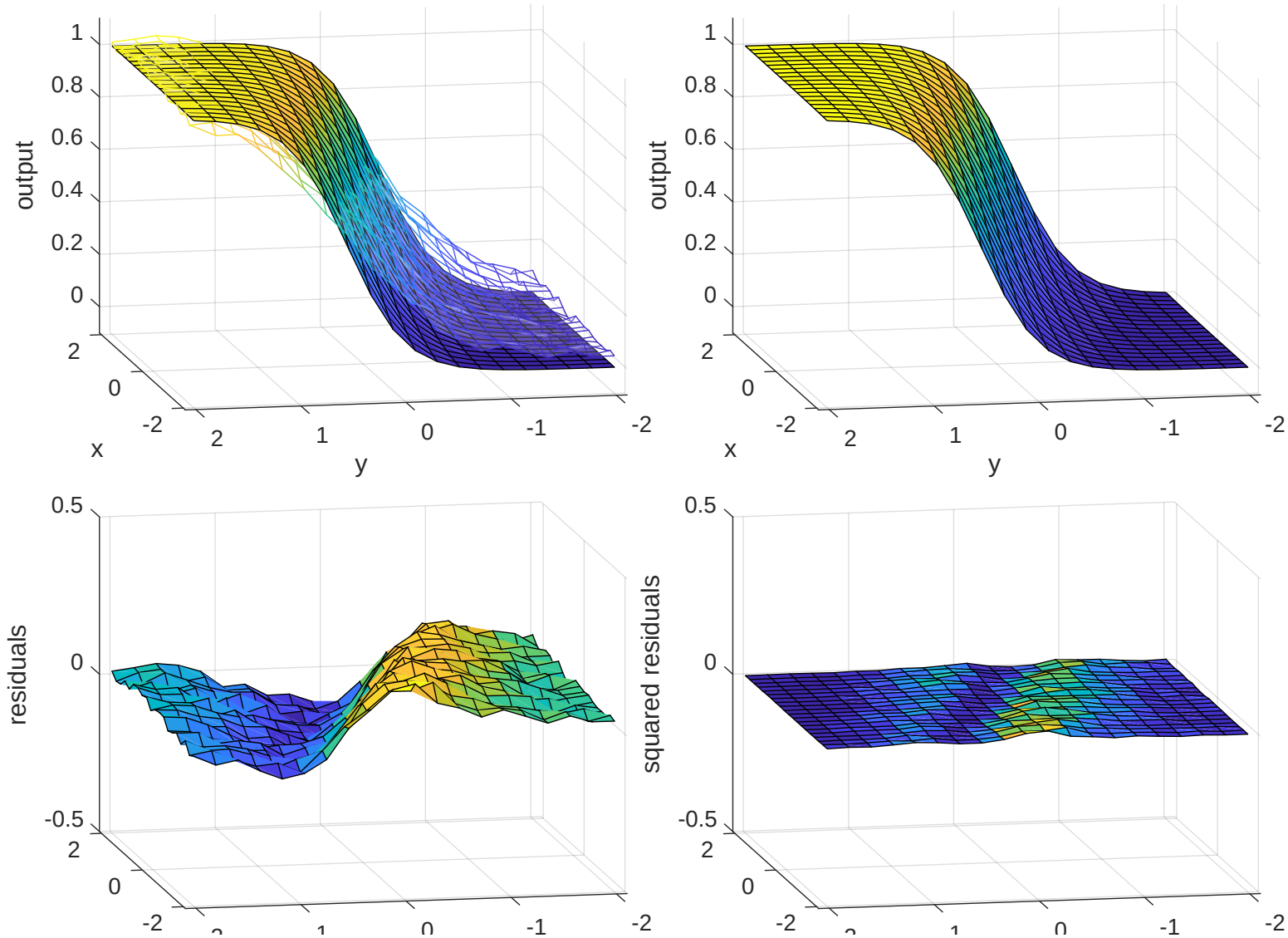
Epoch 28, SSR $= 3.76$



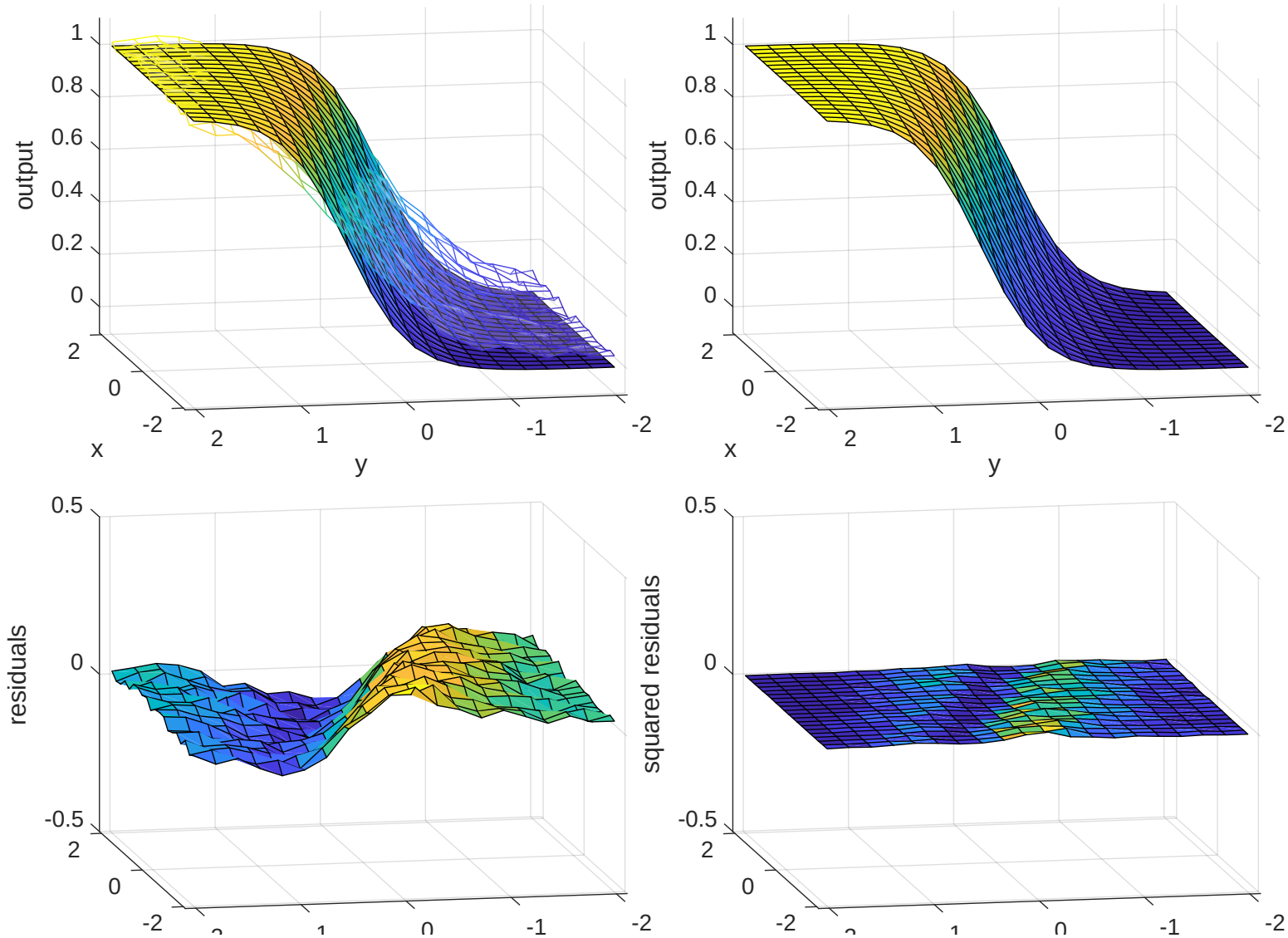
Epoch 30, SSR $= 3.42$



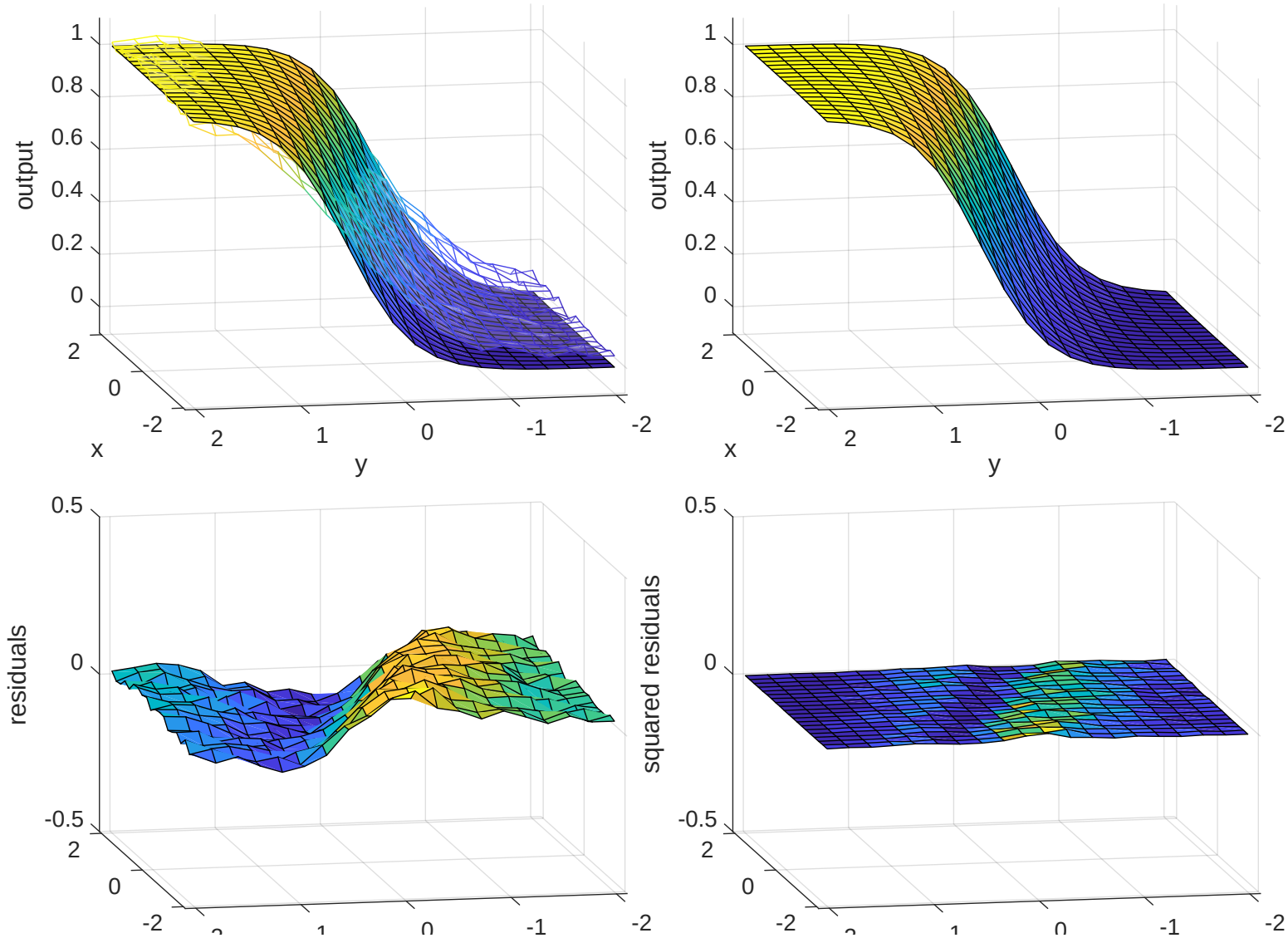
Epoch 32, SSR $= 3.07$



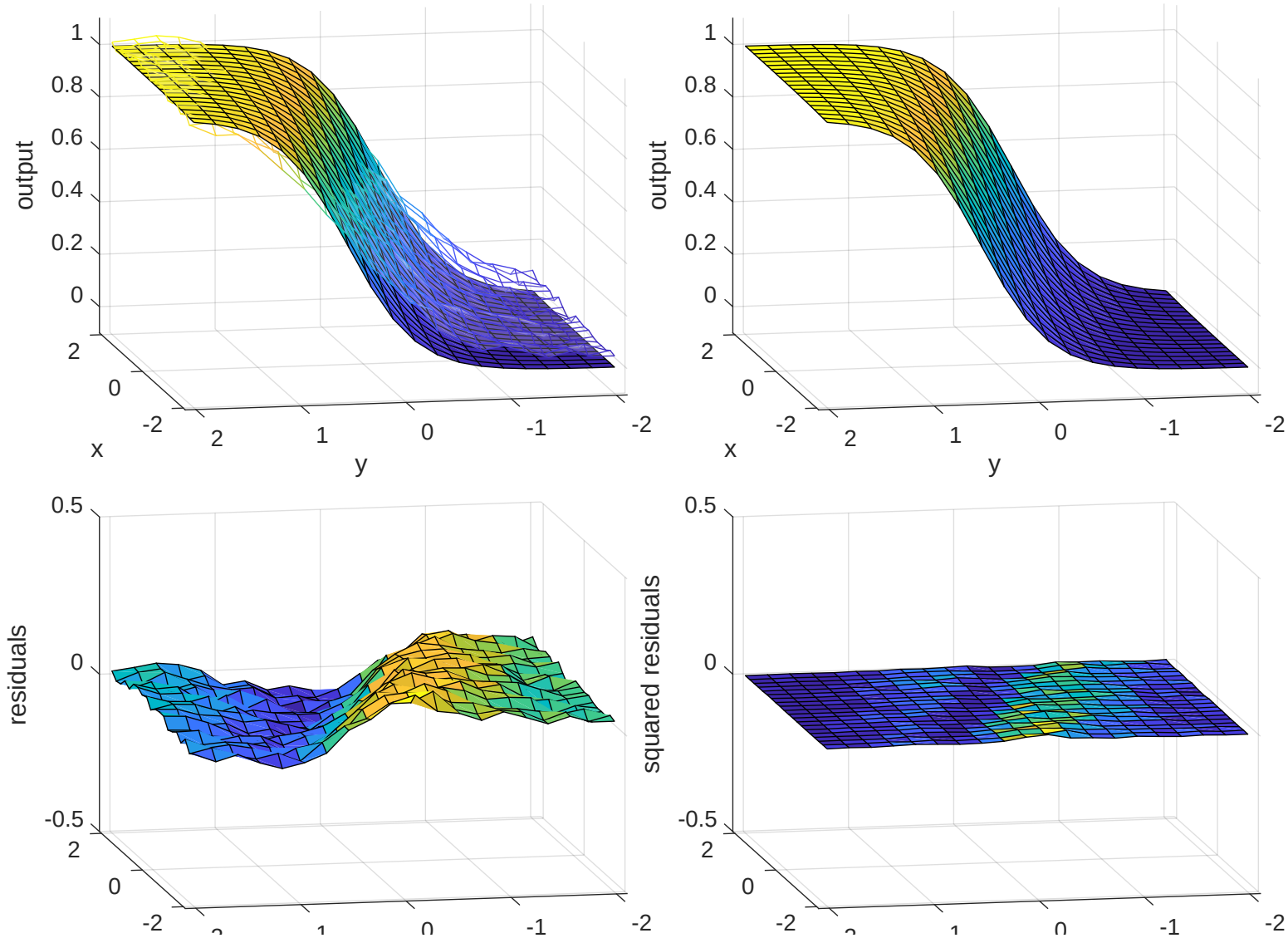
Epoch 34, SSR $= 2.7$



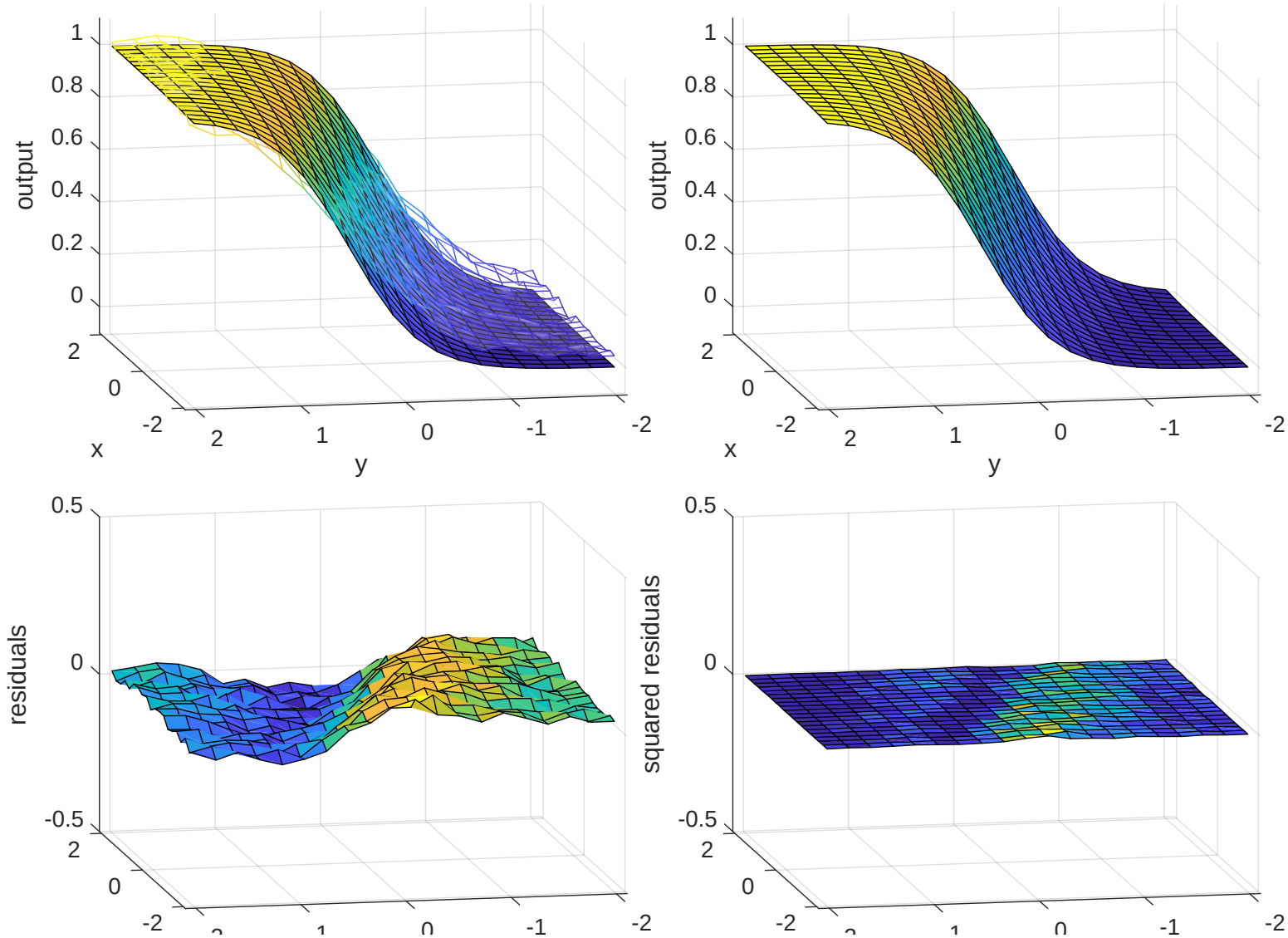
Epoch 36, SSR $= 2.33$



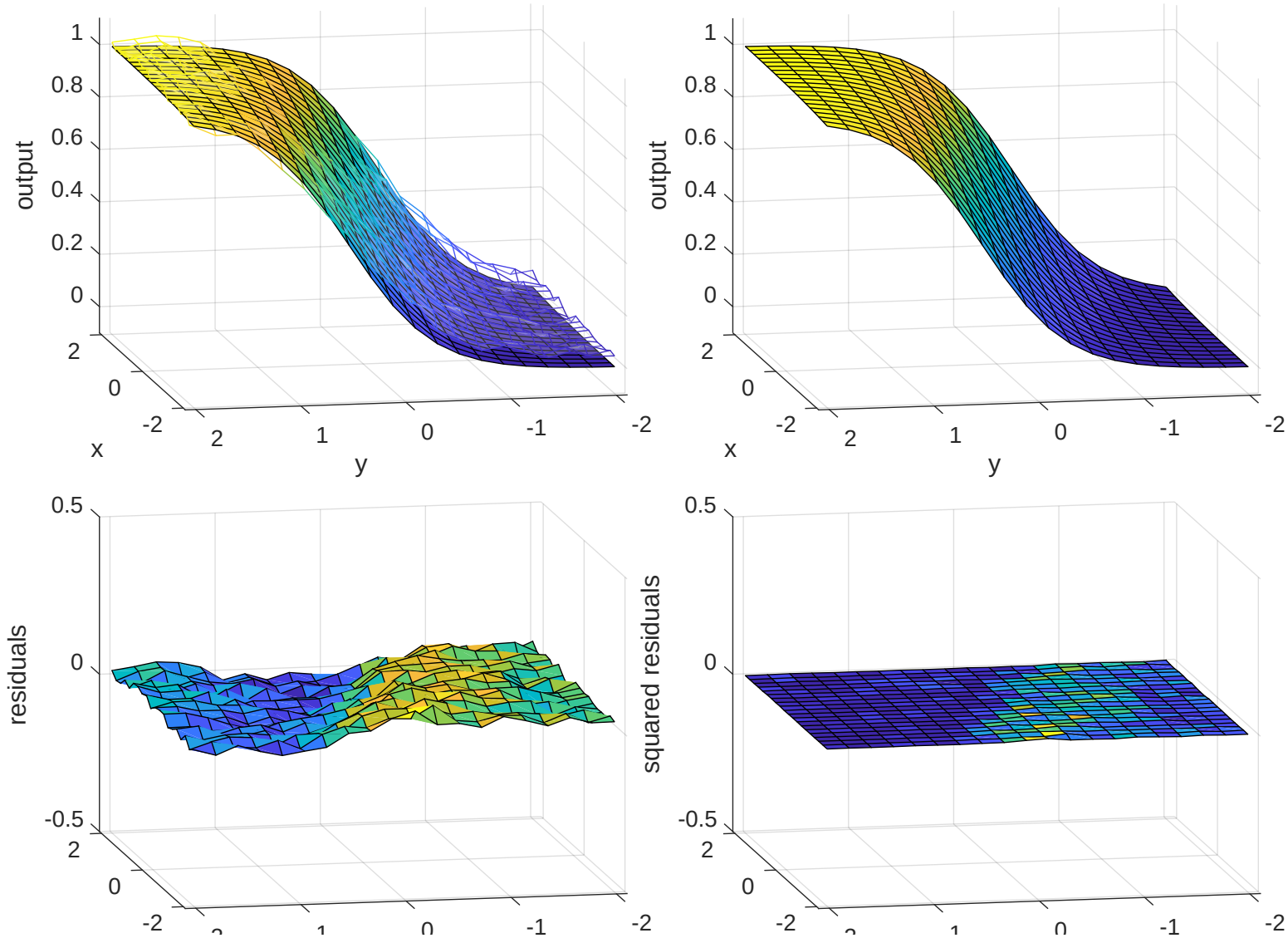
Epoch 38, SSR = 1.97



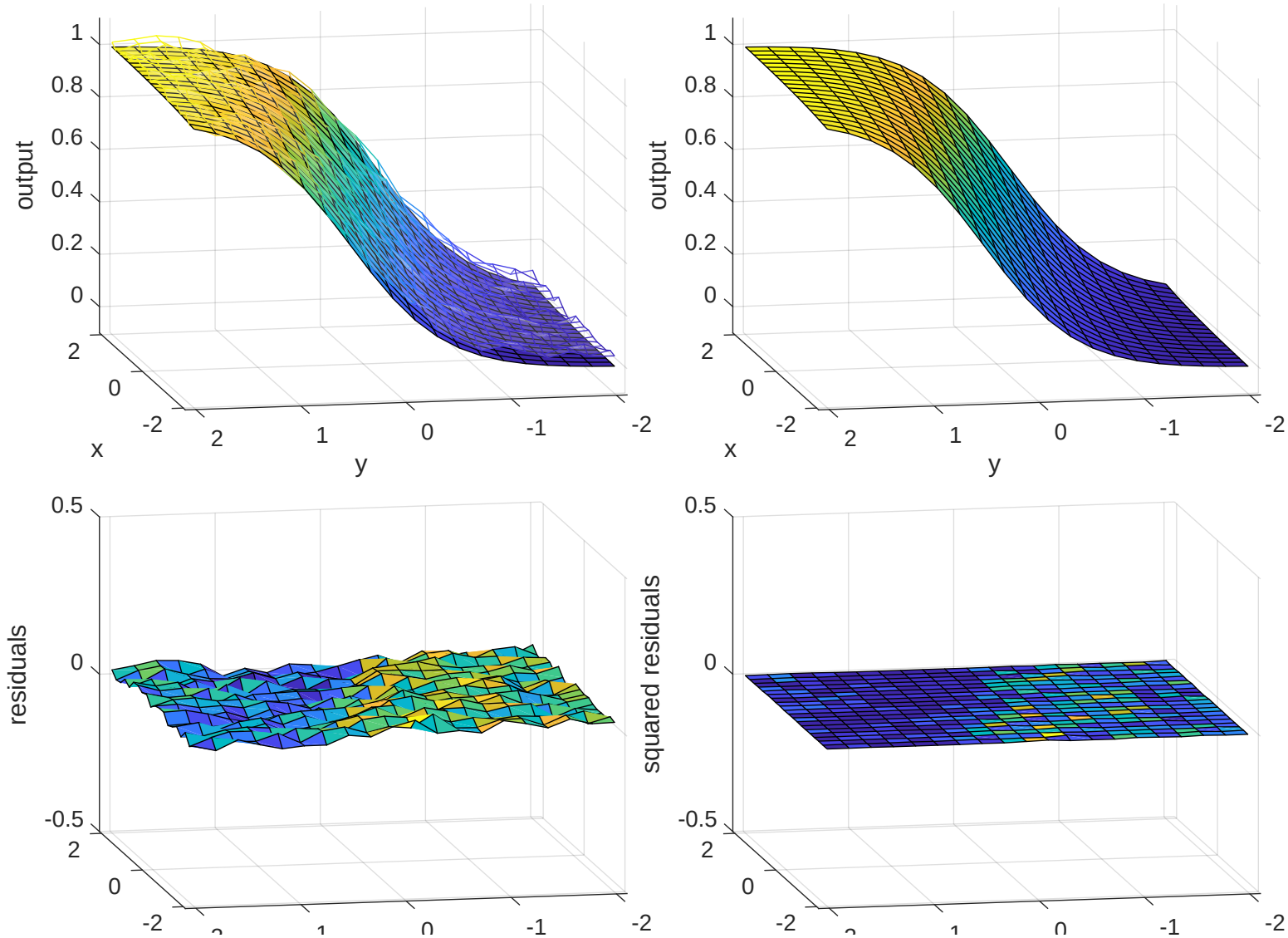
Epoch 40, SSR $= 1.61$



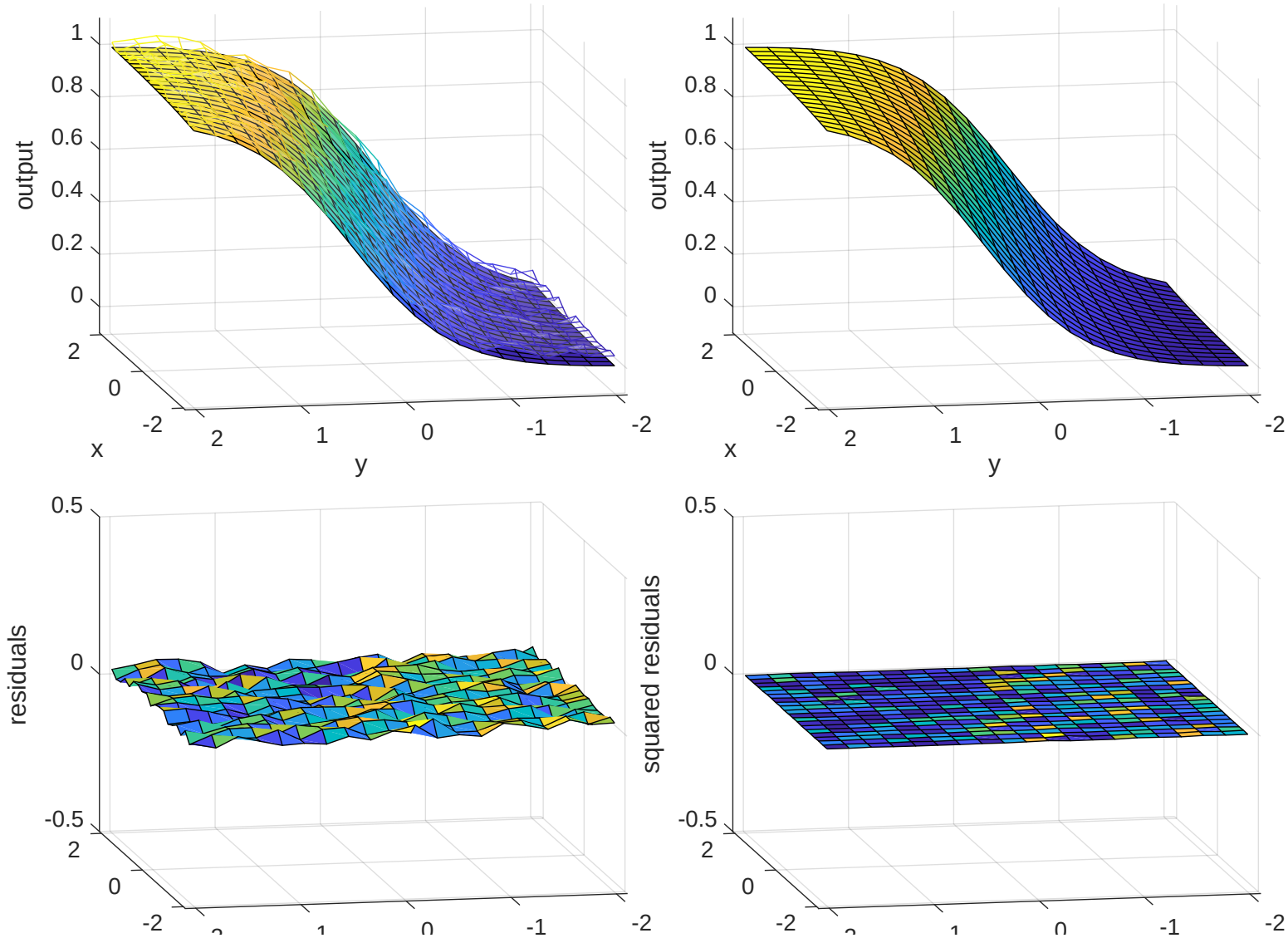
Epoch 45, SSR $= 0.87$



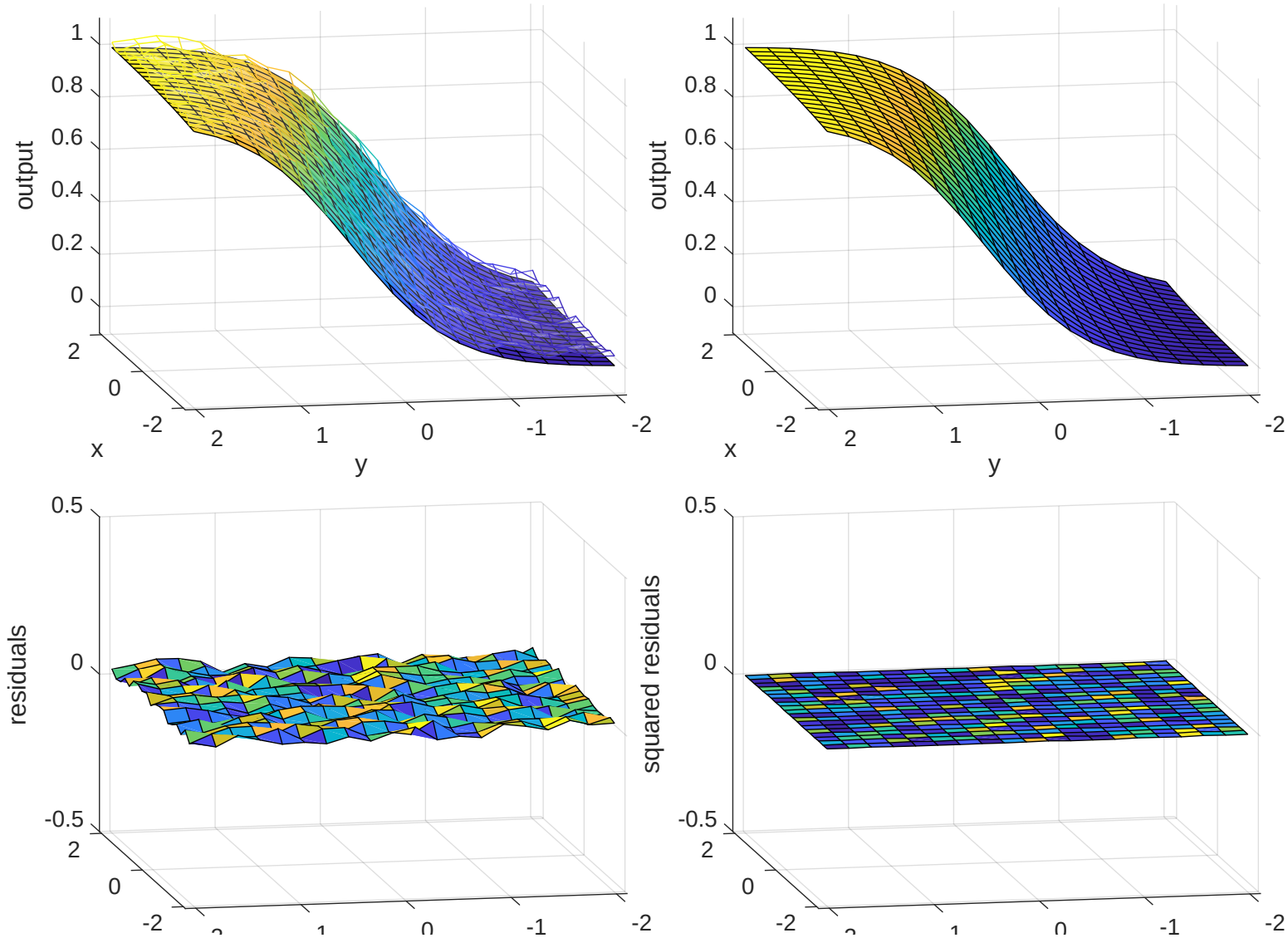
Epoch 50, SSR $= 0.46$



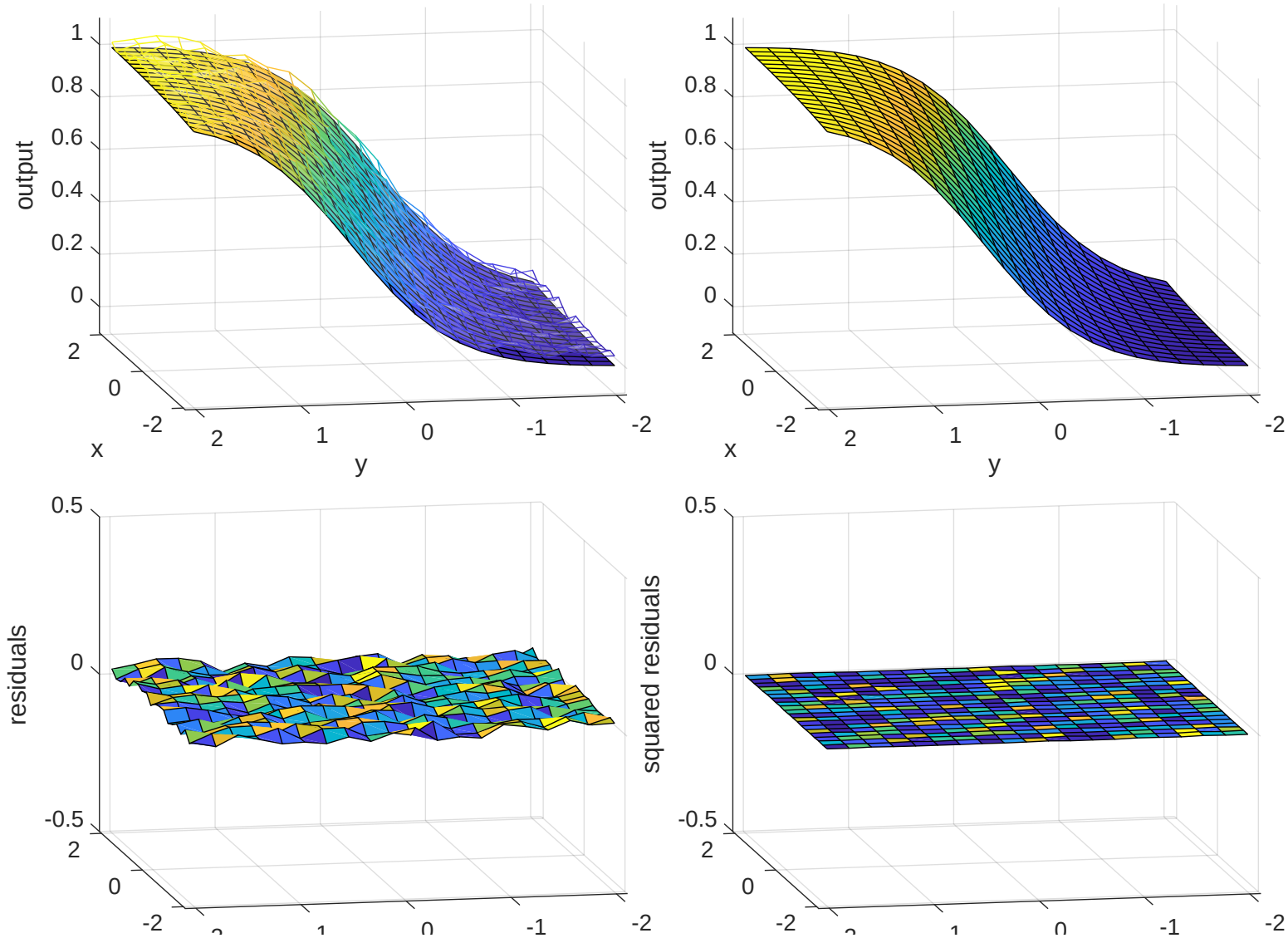
Epoch 55, SSR $= 0.33$



Epoch 60, SSR $= 0.32$

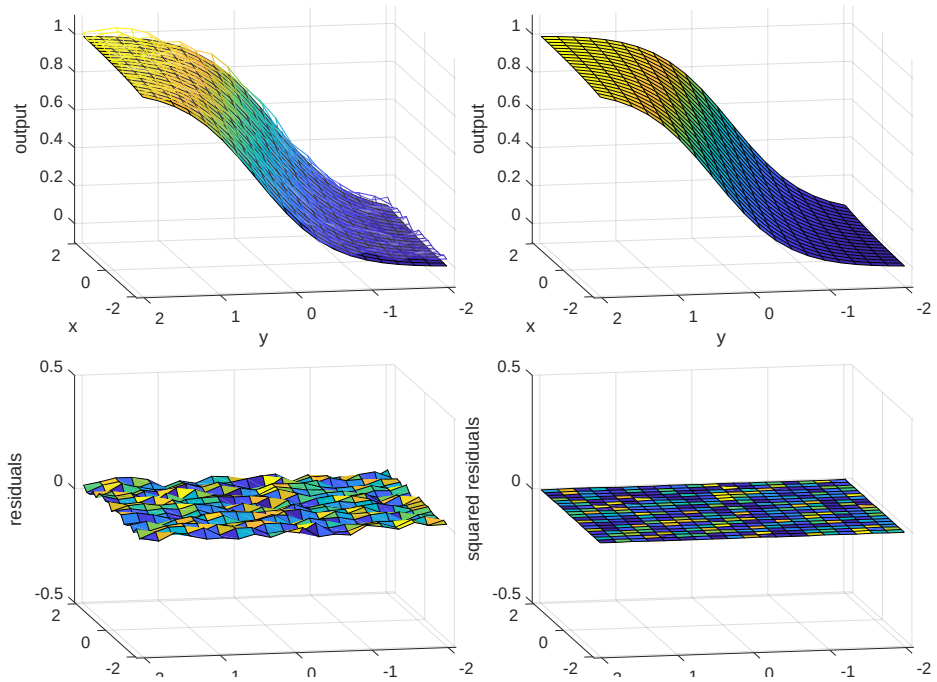


Epoch 65, SSR $= 0.32$



Final result

- Sum of squares of the residuals 1.41592116 at last iteration.
- Weights at last iteration $w_1 = 0.4984$ and $w_2 = 2.00102$.
- Could wone build a better approximation?



Weights evolution