Gradient Descent method in Least Square Regression

Recall our equations:
(i) $\partial t\, S = G_s - D_s * S - k_2 * ER * S + k_{-2} * ERS$
(ii) $\partial t\, ER = k_1 * E * R - k_{-1} * ER - k_2 * ERS + k_{-2} * ERS$
(iii)$\partial t\, ERS = k_2 * ER * S - k_3 * ERS - k_{-2} * ERS$
(iv)$\partial t\, P = k_3 * ERS - D_s * P$
(v) $\partial tF = k_4 * P - D_f * F$

Where ∂t refers to differentiate with respect to t, and the coefficient of reaction rates are denoted k_I, and they are our subject of interest.
The differential equations are non-linear, so an analytical solution may not exist. Fortunately, the idea of bringing in these equations is not for the sake of solving them analytically. Instead, it suffices to dump these equations into any programming software and run a simulation. This alone allows modelling the behaviour of all our components of interest over time. Hence, the major aim of our model is actually the characterization of these coefficients k_{i-}, which will help in the optimization problem eventually. For completing this characterization, we utilized a method called the Gradient Descent Least Square Regression method to determine the best fit for these coefficients.

The idea of the least square regression should be familiar to most scientists: we vary the coefficients in the equation ki to minimize the square distance given by

$$D = \sum_{i=1}^{n}(y_i - p_i)^2$$

However, in our model, there are a total of 9 parameters(coefficients) to tune, meaning that the computer would have to test out changes over 9 dimensions to determine a ‘good enough’ approximation. This is in general, a tedious process, and an optimization is required for the regression to run in an acceptable period.
To explain the regression method, consider the space spanned by our 9 parameters in the model. We have 9 independent parameters that can be tuned, hence we have 9 independent ways of altering the parameters. Mathematically, we say that the space spanned by these parameters is a 9-dimensional space. And let’s add one more dimension to it: the value of the square distance between the data and the model. Recall the destination we want to find, is the set of coefficients that minimizes the square distance. Hence, we’re looking for a valley, and we’re interested in the point in 9-dimensional space that corresponds to the valley. That is our best estimate for the parameter.

Let’s look at a classical analog. Consider the query: ‘which coordinate in the map points to the lowest point within a 100-mile radius of Mt. Everest?’ The answer is simple: simply look at the map for the lowest point you can find near Mt. Everest, and extract their coordinates. Indeed, the least square regression is asking the same question: Which point in the 9-dimensional space points to the valley we’re interested in? However, unlike the classical analog, generating the complete map is an extremely cumbersome task. Assume the sensitivity of our map is 0.01mile. In 2-D maps, we will need to compare $(10000)^2$ points. Well, in 9-D, we will need to compute and compare the square distance for $10^{4*9} = 10^{36}$ points. In addition, our models do not

have an upper bound for the values of the coefficients k_i. The boundaries are 0 and infinite. Hence, to generate the exact valley would require us to test for an infinite number of points.
In practice, it is often impossible to find the exact valley without a given (closed) form of the solution. Often, an approximate solution is good enough: we find a valley deep enough that the difference can be ruled out as noise. And fortunately, this approximate solution is way easier to find and implement: simply move downhill.

By definition, a point in space is a minima (valley) if and only if the derivatives (slopes) over all other dimensions is 0. In other words,

$$\partial x F = \partial y F = \partial z F = \ldots = 0$$

and, from the definition of gradient:

$$\partial x F = \lim_{\delta \to 0} \frac{F(x+\delta, y, z, \ldots) - F(x-\delta, y, z, \ldots)}{2\delta}$$

In our 2-D map analogy, the argument is simple: if the slope in a direction is non-zero, the point would be a trench at best. Existence of a valley requires the slopes in all directions to be zero. Hence, by constantly moving downhill, in the direction of the gradient, we will come to a stop if and only if we hit a valley.

By construction of our differential equations, the gradients of our distance function exists and has to be continuous in the period of the experiment if the solution exist. Then, we can always construct a small neighbourhood around our valley, and gradients in opposite directions from the valley will have opposite parity. Using this property, we can detect the existence of valleys when our gradient changes parity. Following the algorithm, we will eventually reach a point with a small enough gradient—and it suffices if the distance function there is small enough.
The algorithm to continually move downhill allows us to find a local minima without considering all points in our 9-D coefficient space. In fact, we only evaluate our model equations and square distance at the points we're traversing. Ergo, this optimization saves time significantly in least square regression.