Hindawi Publishing Corporation
Mathematical Problems in Engineering
Volume 2011, Article ID 570509, 15 pages
doi:10.1155/2011/570509

Research Article

An Algorithm for Optimally Fitting a Wiener Model

Lucas P. Beverlin,[1] Derrick K. Rollins,[2, 3] Nisarg Vyas,[4] and David Andre[4]

[1] *Intel Corporation, Hillsboro, OR 97124, USA*
[2] *Department of Statistics, Iowa State University, Ames, IA 50010, USA*
[3] *Department of Chemical & Biological Engineering, Iowa State University, Ames, IA 50010, USA*
[4] *BodyMedia Inc., 420 Fort Duquesne Boulevard, Pittsburgh, PA 15222, USA*

Correspondence should be addressed to Lucas P. Beverlin, lucas.p.beverlin@intel.com

Received 2 September 2010; Accepted 15 September 2011

Academic Editor: Elbert Macau

The purpose of this work is to present a new methodology for fitting Wiener networks to datasets with a large number of variables. Wiener networks have the ability to model a wide range of data types, and their structures can yield parameters with phenomenological meaning. There are several challenges to fitting such a model: model stiffness, the nonlinear nature of a Wiener network, possible overfitting, and the large number of parameters inherent with large input sets. This work describes a methodology to overcome these challenges by using several iterative algorithms under supervised learning and fitting subsets of the parameters at a time. This methodology is applied to Wiener networks that are used to predict blood glucose concentrations. The predictions of validation sets from models fit to four subjects using this methodology yielded a higher correlation between observed and predicted observations than other algorithms, including the Gauss-Newton and Levenberg-Marquardt algorithms.

1. Introduction

Wiener networks are widely used in modeling complex nonlinear systems. These networks have the ability to model a wide range of data types, such as gas concentrations [1], blood glucose concentrations [2], and pH levels [3], and their structure can yield parameters with phenomenological meaning [2]. In this work, Wiener networks are used to first convert inputs into their corresponding dynamic responses and then to pass these dynamic responses through a second-order linear regression function to obtain the fitted output response. The parameters needed to convert the inputs into dynamic responses are referred to as dynamic parameters and the parameters of the regression function as static parameters. However, estimating these parameters can be quite challenging, as the behavior of these networks can be highly nonlinear in the dynamic parameters, and the number of parameters, which can be large, increases rapidly as inputs are added. Overfitting can also be a major issue. Given that Wiener networks utilize differential equations for the conversion to dynamic responses, stiffness, which is the situation where the derivative increases or decreases rapidly while the solution to the differential equation does not [4], is another concern. This phenomenon causes an algorithm to take very small steps (i.e., progress slowly) in order to reach an optimal solution. Another issue is that the process being modeled could change over time due to degradation of equipment, an increase in production, or one of many other reasons. While this change could be very gradual, this implies that eventually the fitted model can degrade over time. If this process is online, then a new model will need to be found expediently to minimize downtime and to take into account the new conditions and inputs.

The basic purpose of this article is to present a new methodology for fitting Wiener networks to large input datasets which can overcome these challenges. In the process control literature this is called "process identification." By fitting subsets of the parameters iteratively, we can deal with a large number of parameters and their nonlinearity, as numerical instability in the next iteration is less likely when fitting a subset of the parameters. During optimization, parameter step size is controlled by the value of the objective function, which deals with stiffness. To avoid overfitting, we will utilize what is called "supervised learning" in the statistics literature [5]. In supervised learning, the dataset is broken up into three subsets: training, validation, and test. The model is fit to the training set with the validation set used to determine the number of iterations to use to guard against overfitting. The test set is scrutinized at the end of the optimization process to further evaluate if overfitting has occurred. While our methodology does not perform the optimization as fast as some of the other popular algorithms, utilizing parallel computing has sped up the optimization of a Wiener network using our methodology by roughly 25% in MATLAB [6].

There are other methods for fitting Wiener networks. Due to the Wiener network's nonlinear nature, these are iterative techniques. Note that a nonlinear modeling problem is one with unknown parameters that are functionally nonlinear. Here the objective is to obtain a set of parameters that minimize the sum of squared residuals (i.e., the least squares objective function). Popular techniques for this optimization objective include the Gauss-Newton algorithm and the Levenberg-Marquardt algorithm [7]. Many mathematics/statistics software packages, such as Minitab (Minitab, Inc., State College, PA.), R [8], SAS (SAS Institute Inc., Cary, NC), and MATLAB can implement both of these algorithms. The Levenberg-Marquardt algorithm, which is given in detail in the appendix, is a compromise between the Gauss-Newton algorithm and the method of steepest descent. We have found that fitting all parameters simultaneously using either algorithm can result in overfitting. Even fitting subsets of parameters with just one algorithm has resulted in a model that performs worse than our methodology. Given that the Wiener network used here has a conditionally linear structure since fixing the dynamic parameters yields a linear model, another potential approach is that of Barham and Drane [9]. They fit four different models to argue that alternating between using the usual least squares formula for estimating the linear parameters and a modification of the Gauss-Newton algorithm suggested by Hartley [10] for the nonlinear parameters will generally perform better than either Hartley's modified Gauss-Newton algorithm or the Levenberg-Marquardt algorithm alone. However, using least squares to fit the linear parameters tended to badly overfit the model irrespective of the dynamic parameters. A final method we reviewed in this work is the GRG2 algorithm [11], which is utilized in the Solver program in Microsoft Excel. This was used to fit the Wiener networks used to model blood glucose concentrations in [2]. It attempts to fit models using a generalized reduced gradient algorithm, which can handle constraints on parameters and

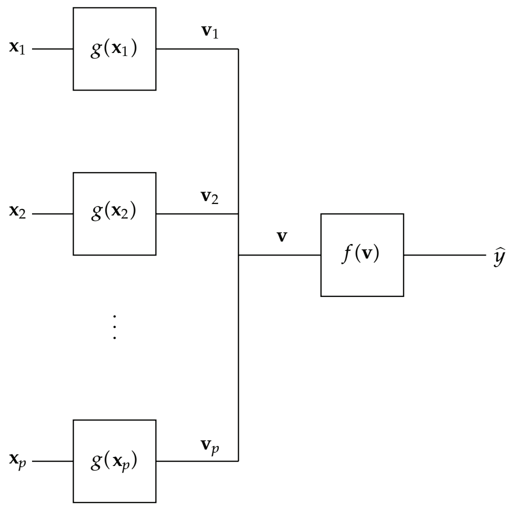

Figure 1: A graphical representation of the Wiener model.

is found to be fairly robust. However, for supervised learning with this implementation in Excel, the algorithm must be paused after each iteration for inspection, which becomes very time consuming if there are more than a few parameters in the model.

The proposed methodology is presented with the following outline. First a detailed description of the Wiener network for multiple inputs and a single output is given to establish the problem context. After this section the details of the methodology and an example are given to illustrate the algorithm in the fourth section. Finally concluding remarks and some ideas for future work are given in the last section.

2. The Wiener Network

In this section, a detailed description of a Wiener network is given to establish the context of the problem. These networks have a powerful structure for modeling nonlinear dynamic systems. A block diagram with p inputs and one output is given in Figure 1.

As shown in Figure 1, each input $\mathbf{x}_i$ is first passed through a dynamic linear block, denoted $g(\mathbf{x}_i)$ and converted into its corresponding dynamic variable $\mathbf{v}_i$. Following Rollins et al. [2], we will use the following second-order plus-lead plus-dead-time differential equation:

$$\tau_i^2(t,X)\frac{d^2v_i}{dt^2}(t) + 2\tau_i(t,X)\zeta_i(t,X)\frac{dv_i}{dt}(t) + v_i(t,X) = \tau_{ai}(t,X)\frac{dx_i}{dt}(t-\theta_i) + x_i(t-\theta_i), \qquad (2.1)$$

where τ_i is a time constant, τ_{ai} is a lead parameter, ζ_i is a damping coefficient, and θ_i denotes dead time. For simplicity, we will assume that the dynamic parameters are time and space invariant, that is, $\tau_i(t,X) = \tau_i$, $\tau_{ai}(t,X) = \tau_{ai}$, and $\zeta_i(t,X) = \zeta_i$, and for the rest of this section, fix $\theta_i = 0$. Also when referring to $v_i(t,X)$, we will write $v_i(t)$ henceforth. We will use $\boldsymbol{\tau}$, $\boldsymbol{\tau}_a$, and $\boldsymbol{\zeta}$ to denote all time constants, lead parameters, and damping coefficients, respectively.

To find a recursive definition for $\mathbf{v}_i$, a forward difference approximation to $(dv_i/dt)(t)$ will be used. First let $j = t/\Delta t$ and $d_i = \theta_i/\Delta t$ so that $v_i(t) = v_{ij}$ and $x_i(t - \theta_i) = x_{i,j-d_i}$. Thus,

$$\frac{dv_{ij}}{dt} \approx \frac{v_i(t) - v_i(t - \Delta t)}{\Delta t} = \frac{v_{ij} - v_{i,j-1}}{\Delta t}, \tag{2.2}$$

$$\frac{d^2 v_{ij}}{dt^2} \approx \frac{(dv_{ij}/dt) - (dv_{i,j-1}/dt)}{\Delta t} \tag{2.3}$$

$$\approx \frac{((v_{ij} - v_{i,j-1})/\Delta t) - ((v_{i,j-1} - v_{i,j-2})/\Delta t)}{\Delta t} \tag{2.4}$$

$$\approx \frac{v_{ij} - 2v_{i,j-1} + v_{i,j-2}}{\Delta t^2}. \tag{2.5}$$

By substituting (2.2) and (2.5) into (2.1),

$$\begin{aligned} v_{ij} = {} & \frac{2\tau^2 + 2\tau\zeta\Delta t}{\tau^2 + 2\tau\zeta\Delta t + \Delta t^2} v_{i,j-1} - \frac{\tau^2}{\tau^2 + 2\tau\zeta\Delta t + \Delta t^2} v_{i,j-2} + \frac{\Delta t(\tau_a + \Delta t)}{\tau^2 + 2\tau\zeta\Delta t + \Delta t^2} x_{i,j-d_i} \\ & - \frac{\tau_a \Delta t}{\tau^2 + 2\tau\zeta\Delta t + \Delta t^2} x_{i,j-d_i-1}. \end{aligned} \tag{2.6}$$

Next all of these dynamic variables are passed through a static nonlinear block, denoted $f(\mathbf{v})$ in Figure 1, in order to obtain the predicted value of the response variable at time t. Following Rollins et al. [2], we use a second-order regression function with linear terms, quadratic terms, and second-order interaction terms, giving

$$y_j = a_0(t, X) + \sum_{i=1}^{p} a_i(t, X) v_{ij} + \sum_{i=1}^{p} b_i(t, X) v_{ij}^2 + \sum_{i=1}^{p-1} \sum_{k=i+1}^{p} c_{ik}(t, X) v_{ij} v_{kj} + \epsilon_j, \tag{2.7}$$

where ϵ_j is a normally distributed error term with mean 0 and variance σ^2 and that for any $k \neq j$, ϵ_j and ϵ_k are independent. Again assume that the parameters are invariant with respect to time and space, for example, $a_0(t, X) = a_0$. The vector of parameters corresponding to the linear terms will be denoted by $\mathbf{a}$, the quadratic terms by $\mathbf{b}$, and the interaction terms by $\mathbf{c}$.

3. The Proposed Parameter Estimation Algorithm

In this section the featured algorithm we are proposing to solve the nonlinear regression problem given in the previous section will be described. Following Rollins et al. [2], the objective of this modeling problem is to maximize the true but unknown correlation coefficient between the measured and fitted glucose concentrations that is defined by $\rho_{y,\hat{y}}$ and estimated by r_{fit}. More specifically, under this objective a model is declared *useful* if and only if

$$\rho_{y,\hat{y}} > 0. \tag{3.1}$$

The meaning of this criterion is that predictions of blood glucose concentrations from the model decrease and increase with measured blood glucose concentrations beyond some degree of mere chance; that is, there is true positive correlation. Notwithstanding, the closer this value is to the upper limit of 1, the more useful the model. Therefore, to achieve this objective, one seeks to identify a model with a sufficiently large value of r_{fit}. To this end, the data are separated into a set for training and a set for validation and/or testing. The training set is used to build the model, and the validation (or testing) set is used to evaluate the model against data that were not directly used by the optimization process to estimate the model parameters. However, due to the highly complex mapping of the parameters into the response space of r_{fit}, the following indirect criterion is used:

$$\begin{aligned} &\text{Maximize} \quad r_{\text{fit}} \text{ by minimizing } \underset{\boldsymbol{\Theta}}{\text{SSE}} = \sum_{i=1}^{n} \left(y_i - \hat{y}_i\right)^2 \\ &\text{Subject to :} \quad \zeta_i > 0,\ \tau_i > 0,\ \theta_i \geq 0\ \forall i, \end{aligned} \tag{3.2}$$

where $\boldsymbol{\Theta}$ is a vector representing the estimated dynamic and static parameters $\boldsymbol{\tau}, \boldsymbol{\zeta}, \boldsymbol{\tau}_a, \boldsymbol{\theta}, \mathbf{a}, \mathbf{b}, \mathbf{c}$ and n is the number of observations in the training set. The objective criterion is used under the assumption that minimizing SSE is equivalent to maximizing r_{fit}. While there is no formal proof for this assumption, experimental evidence supports a strong tendency for this relationship [2].

A model that is nonlinear in parameters, such as the proposed structure, does not have the condition that that sum of the residuals equal 0 as in the case of linear regression. However, under (3.2), the sum of the residuals in the training set should be small and thus a secondary criterion on the closeness of y_i and $\hat{y}_i$ for accuracy can be used. This measure of accuracy, denoted the average absolute error (AAE), is defined as

$$\text{AAE} = \frac{\sum_{j=i_{\text{init}}}^{i_{\text{fin}}} \left|y_j - \hat{y}_j\right|}{i_{\text{fin}} - i_{\text{init}}}, \tag{3.3}$$

where i_{init} is the initial observation used for calculation of this statistic and i_{fin} is the final observation used. Hence accuracy is judged to increase as AAE decreases.

Thus, in addition to sufficiently large r_{fit} values for both the training and test/validation datasets, an acceptable model must also have a relatively small value of AAE in training. This secondary criterion is not imposed in testing/validation because (3.2) forces small residuals for training data only. Furthermore, if a model is capable of a high r_{fit} as demonstrated in training then high accuracy can be obtained with effective feedback correction or feedback control to reduce or eliminate bias.

To obtain a useful model the proposed method fits a subset of the parameters using an iterative approach, and the validation set is used to terminate optimization to guard against overfitting. As mentioned before, this was done because attempts to fit with respect to every

```
$score = .1 \cdot r_{\text{fit}}/.8 + .9 \cdot r_{\text{Val}}/.8$
for $i = 1$ to 11 do
    repeat
        $scoreold = score$
        $a_i = a_i + .2$
        Update $score$
    untill $score < scoreold$
    if $a_i = 0$ then
        Replace addition operation with subtraction operation and repeat the loop.
    end if
end for
```

Algorithm 1: Starting algorithm.

parameter at once tend to overfit the training set. To determine which iteration is best, each iteration was given a score based on a linear combination of two statistics whose maximum is 1: r_{fit} and the correlation r_{Val} between observed and predicted response values in the validation set. To establish notation, one can write the correlation between datasets $\{x_i\}_{i=1}^{K}$ and $\{t_i\}_{i=1}^{K}$ as

$$r_{\text{Val}} = \frac{\sum_{i=1}^{K}(x_i - \overline{x})\left(t_i - \overline{t}\right)}{\sqrt{\sum_{i=1}^{K}(x_i - \overline{x})^2 \sum_{i=1}^{K}\left(t_i - \overline{t}\right)^2}},$$
$$\overline{x} = \frac{1}{K}\sum_{i=1}^{K} x_i, \tag{3.4}$$
$$\overline{t} = \frac{1}{K}\sum_{i=1}^{K} t_i,$$

where K is the total number of observations in the dataset. It was used in order to ensure that the predictions in the validation set are properly tracking changes found in the actual data.

After setting the starting values for the parameters, we then execute Algorithm 1 of our methodology. Note that we divide r_{fit} by .9 and r_{Val} by .8 before applying the coefficients. This was done because in practice these values appear to be the maximum attainable values for r_{fit} and r_{Val} in the example in the proceeding section. These values can be changed if one has a rough guess as to what the maximum attainable values are. The score is then calculated as $.1 \cdot r_{\text{fit}}/.8 + .9 \cdot r_{\text{Val}}/.8$. Starting with a_1, we will successively add 0.2 until the score decreases. We then subtract 0.2 in order to retain the maximum score. This is done for each a_i. If choosing $a_i = 0.2$ yields a lower score than setting $a_i = 0$, then we successively subtract 0.2 from a_i until the score decreases and readd 0.2 once a decrease is observed.

We then attempt to fit a model using all parameters with the Levenberg-Marquardt and Gauss-Newton algorithms [7]. The Levenberg-Marquardt algorithm is given in the appendix, as the Gauss-Newton algorithm used here is simply the Levenberg-Marquardt algorithm where $\alpha = 0$. The Levenberg-Marquardt algorithm is fit nine times, each with the same starting parameters. The only difference between these trials is the values of the damping parameter α used, which were 100, 10, 1, .01, .0001, $1e-6$, $1e-8$, $1e-10$, and 0.

Note that α affects the step size of each iteration, as each model responds differently to each value of α chosen. For each iteration of each trial, a score is calculated as before, and the parameter estimates corresponding to the iteration with the highest score that satisfies the parameter constraints among these trials are returned at the end of this loop.

The next stage of this methodology takes the parameter estimates from the previous stage and proceeds to fit subsets of these parameters at a time. Since using one algorithm exclusively has generally yielded weaker results, we chose to use three different algorithms and compare their results. The first algorithm that is attempted by our algorithm is the BFGS [7] algorithm. This is a quasi-Newton method in that it approximates the Hessian, which is used in Newton's method, with a rank 2 matrix that depends on the Jacobian. Note that we remove the difficulty of finding the inverse of a matrix by using the Sherman-Morrison-Woodbury formula [12]. We also used a trust-region version of this algorithm, in that we fixed the maximum step size of this algorithm. However, it is not guaranteed that a step from this algorithm will result in a decrease in the objective function. Hence the second algorithm we will use is the conjugate gradient [13] algorithm. This algorithm requires very little storage and is based on the idea of conjugate directions, although these directions lose conjugacy if the model is not well approximated by a quadratic approximation. However, the conjugate gradient algorithm tends to take a large step followed by several small steps, and these large steps tend to overfit when fitting Wiener networks. On top of this, the derivatives of the objective function with respect to the dynamic parameters must be approximated and can be at times unreliable. Thus the final algorithm we use is the Nelder-Mead [12] algorithm. This algorithm first generates $n + 1$ points equidistant from one another and from the starting values, which is called a simplex. It then uses function evaluations to move the simplex toward a minimum as well as to expand or shrink the simplex. Thus it uses more function evaluations in place of derivatives. Details on these algorithms are given in the appendix. Note that to run the Nelder-Mead algorithm, the built-in function `fminsearch` in MATLAB was used. The number of iterations that were ran for the Nelder-Mead algorithm per iteration of this stage and the trust-region radius of the BFGS algorithm was chosen based on the value of r_{fit} at the start of each respective algorithm.

As each iteration of each algorithm is determined from a subset of the parameters, once again a score is calculated, and the parameter estimates with the highest score where r_{fit} has increased over the value found from the starting parameters and each constraint is satisfied are chosen as the new parameter estimates. There are four methods of choosing the subsets in our methodology. For the first method, we fit $\mathbf{a}$ first, then $\mathbf{b}$, $\mathbf{c}$, $\boldsymbol{\tau}$, $\boldsymbol{\zeta}$, and finally $\boldsymbol{\tau}_a$. This is repeated if the score of the parameter estimates $\widehat{\boldsymbol{\theta}}$ has improved by at least 0.0001 from the score of the parameter estimates used as starting values to fit $\mathbf{a}$ up to three times. For the second method, the first subset of parameters to be fit is the set of static parameters that depend on the first input, that is, $\{a_1, b_1, c_{12}, c_{13}, \ldots, c_{1,11}\}$. The second subset of parameters is the set of static parameters that depend on the second input, that is, $\{a_2, b_2, c_{12}, c_{23}, c_{24}, \ldots, c_{2,11}\}$. Since we have eleven inputs, there will be eleven such subsets. The final three subsets of parameters to be fit are $\boldsymbol{\tau}, \boldsymbol{\zeta}$, and $\boldsymbol{\tau}_a$. Again this is repeated up to three times if a score increase of at least 0.0001 is observed each time. For the third method, we will fit the same subsets of static parameters as in stage two. However, we will use different subsets of the dynamic parameters. We first fit three subsets of $\boldsymbol{\tau}$ in this order: $\{\tau_1, \tau_2, \tau_3\}$, $\{\tau_4, \tau_5, \tau_6, \tau_7\}$, $\{\tau_8, \tau_9, \tau_{10}, \tau_{11}\}$. Then the corresponding subsets of $\boldsymbol{\zeta}$ are fit in the same order, followed by these subsets of $\boldsymbol{\tau}_a$. For the last method, the first subsets of parameters fit are $\{a_1, b_1\}, \{a_2, b_2\}, \ldots, \{a_{11}, b_{11}\}$. Next we fit subsets of parameters corresponding to the interaction terms. First we fit the interaction parameters corresponding to the first input,

$\{c_{12}, c_{13}, \ldots, c_{1,11}\}$, followed by the second input, $\{c_{12}, c_{23}, c_{24}, \ldots, c_{2,11}\}$, and so on. After these parameters we fit the same subsets of dynamic parameters as in the previous stage. Assuming the value of r_{fit} has increased by 0.002 since the start of the first stage, we will update the coefficients for calculating the scores and the algorithm will return to the first stage. If not, then we attempt to fit a model one parameter at a time. If after cycling through every parameter three times the value of r_{fit} has not increased from the first stage by 0.001, then the algorithm exits. Otherwise the coefficients are updated, and the algorithm returns to the first stage. After two iterations through each of the four methods, each stage may only be repeated twice instead of three times.

Finally we discuss how to update the coefficients of the score. For the first iteration of this methodology, the score is calculated as before: $.1 \cdot r_{\text{fit}}/.9 + .9 \cdot r_{\text{Val}}/.8$. To update the coefficients, first let $w = r_{\text{fit}} + 4r_{\text{Val}}$. Then they are updated to be r_{fit}/w, and $4r_{\text{Val}}/w$, respectively. This is done to force the correlation between the predicted and observed values in the validation set to be weighted heavily. This can be altered depending on how important it is to the researcher to achieve a high r_{Val}. The weight of r_{fit} is large enough so that a large increase can be chosen if a small enough decrease in r_{Val} is observed.

4. An Example: Blood Glucose Concentration Prediction of Type 2 Diabetics

We now illustrate our methodology and compare it to other algorithms mentioned in this paper. In this study, several subjects who exhibit significant variation in their blood glucose concentrations participated in a study in order to determine if their blood glucose concentrations can be adequately predicted from a Wiener network using activity variables, food consumption, and time of day. Since type 2 diabetes affects each subject differently, a model was built for each individual. Due to time constraints to meet the submission deadline, four of the subjects will be evaluated in this work.

In order to obtain blood glucose concentrations, the Medtronic MiniMed Continuous Glucose Monitoring CGMS System Gold (Medtronic Minimed, Northridge, Calif) was used. The SenseWear Pro3 Body Monitoring System (BodyMedia, Inc., Pittsburgh, PA) was used to measure the activity variables used in building this model. From these devices measurements of activity and blood glucose concentrations were obtained every five minutes. Subjects were also asked to record the food that they ingested during this time onto a PDA, which used the Weightmania Pro software (Edward A. Greenwood, Inc., Cambridge, Mass). Other than the necessary downtime to download the data from these devices, data were collected by these devices twenty-four hours a day for four weeks. While the SenseWear Pro3 Body Monitoring System can measure over 30 activity variables, it was decided after much trial and error to use only 7 of these variables for their Wiener network. Of the other four variables, three of them, carbohydrates, fat, and protein, are food variables that measure the amount of each consumed in grams every five minutes. The final one, time of day, allows one to capture the Circadian rhythm of each individual's body [14]. It assumes values from 0, denoting midnight, to 1439, denoting 11:59 pm. A table of all inputs is given in Table 1.

Due to the amount of data available for each subject, the first week of a subject's data was used to fit an individual model for that subject and the subsequent two weeks as a validation set. The starting values were set to be 0 for each a_i, b_i, and c_i other than a_0, which was set to $\overline{y}_{\text{Tr}}$. The dynamic variables were set to parameter estimates found from fitting a pilot model. We have fit models using six different methods. The first method

Table 1: A table of inputs used in this type 2 diabetes study.

Variable type	Variables		
Activity	Transverse accel.—peaks	Energy expenditure	Near body temp.
	Longitudinal accel.—average	Galvanic skin response	Heat flux
	Transverse accel.—MAD		
Food	Carbohydrates	Fat	Protein
Circadian	Time of day		

Table 2: Training and validation statistics for Wiener networks fit to model blood glucose concentrations for four diabetic subjects. Note that PM is the proposed methodology, GN is the modified Gauss-Newton algorithm, LM is the modified Levenberg-Marquardt algorithm, and ES is the Excel Solver methodology. Note that ES was fit manually.

Subject	Algorithm	AAE_{Tr}(mg/dL)	r_{fit}	r_{Val}	Time (s)
1	PM	12.4	0.60	0.59	4127
	GN	12.5	0.40	0.54	347
	LM	12.5	0.45	0.56	83
	ES	7.2	0.83	0.52	—
2	PM	6.8	0.84	0.56	10592
	GN	9.0	0.77	0.43	535
	LM	6.2	0.86	0.58	97
	ES	6.9	0.84	0.49	—
3	PM	11.5	0.71	0.52	4735
	GN	6.8	0.81	0.58	793
	LM	7.2	0.80	0.55	96
	ES	7.8	0.75	0.48	—
4	PM	11.4	0.82	0.68	7032
	GN	11.8	0.81	0.60	1028
	LM	11.7	0.81	0.59	79
	ES	13.3	0.72	0.51	—

was the proposed methodology (PM). The second method utilized the Gauss-Newton (GN) algorithm, and another method used the GN algorithm in a method similar to PM. More specifically, the parameters were fit using the GN algorithm, with the subsets as done in PM. This particular method is called the modified GN algorithm. The fourth method utilized the Levenberg-Marquardt (LM) algorithm, and the fifth method was a modified LM algorithm, where the modifications were made in the same manner as the modified GN algorithm. Finally models were fit using the Excel Solver (ES) routine. Other than the ES routine, all models were fit using MATLAB on a computer with a 2.66 GHz Intel Core 2 Quad processor and 4 GB of RAM. The comparative results are given in Table 2. For each subject, the correlation between predicted and observed blood glucose concentrations in the validation set is at least 0.48. Here we desire a high r_{Val} as we wish to track the actual blood glucose concentration closely, since we do not wish to miss sudden changes in blood glucose concentration, particularly if it becomes very low (<40 mg/dL) due to the health consequences. This is why we chose the coefficients for the score as stated earlier.

We have split the results into two tables. Table 2 compares the methods that fit subsets of the parameters, and Table 3 compares the proposed methodology to the generic GN and

Table 3: Training and validation statistics for Wiener networks fit to model blood glucose concentrations for four diabetic subjects. Note that PM is the proposed methodology, GN is the Gauss-Newton algorithm, and LM is the Levenberg-Marquardt algorithm.

Subject	Algorithm	$\mathrm{AAE}_{\mathrm{Tr}}$ (mg/dL)	r_{fit}	r_{Val}	Time (s)
1	PM	12.4	0.60	0.59	4127
	GN	12.4	0.30	0.53	1.26
	LM	12.5	0.35	0.54	3.36
2	PM	6.8	0.84	0.56	10592
	GN	11.1	0.51	0.23	3.08
	LM	12.8	0.25	0.33	4.66
3	PM	11.5	0.71	0.52	4735
	GN	10.0	0.54	0.37	2.64
	LM	11.2	0.51	0.39	4.09
4	PM	11.4	0.82	0.68	7032
	GN	18.7	0.28	0.39	1.57
	LM	14.7	0.69	0.37	3.68

LM algorithms. Looking at Table 2, we see that the modified GN algorithm had difficulty with subject 1, as r_{fit} for this subject was 0.40. This indicates that $J'J$ is ill conditioned for this subject at the starting values, and since this could happen for other subjects, the Gauss-Newton algorithm alone would not be a good choice for fitting a Wiener network to these subjects. The LM algorithm does not typically have such a difficulty due to its damping parameter α, and the modified LM algorithm generally outperformed the modified GN algorithm. However it should be noted that α was set depending on the value of R^2: it was set to 100 if $r_{\mathrm{fit}} < 0.3$, 1 if $0.3 \leq r_{\mathrm{fit}} < 0.5$, 10^{-3} if $0.5 \leq r_{\mathrm{fit}} < 0.7$, and 10^{-6} if $R^2 \geq 0.7$. This was done in order to alleviate ill conditioning and to allow for more aggressive steps when ill conditioning is no longer a problem. We see that the modified LM and GN algorithms fit much faster than the proposed method, but this is not a major problem since we are fitting these models offline. As for the GRG2 algorithm in Excel, it appears to be very competitive with the proposed method for finding a model with a large r_{fit} value, but the proposed method outperforms this method for every subject's validation set.

As for Table 3, we compare the proposed methodology to fitting models under supervised learning with all parameters simultaneously using either the GN algorithm or the LM algorithm. Of course these other algorithms will fit much faster as there is only one set of parameters to be fit. However, we see that r_{fit} and r_{Val} are greater for each subject when fitting a model with the proposed methodology than either such algorithm.

5. Concluding Remarks

This paper presents a methodology that appears to find better parameter estimates for Wiener networks than other previous algorithms. This methodology uses a score based on two statistics: r_{fit} and r_{Val} in order to avoid overfitting. It also uses a grid search and the Levenberg-Marquardt algorithm in order to improve on the starting parameters. Subsets of the parameters are then fit using the Nelder-Mead, BFGS, and conjugate gradient algorithms in order to overcome issues such as stiffness, poorly approximated derivatives, and nonlinearity. However, we believe there are a few things that can be done to enhance the algorithm.

First, instead of solving the differential equation in order to calculate the dynamic variables used in the example, they were approximated. If possible, one should obtain exact values for the dynamic variables, as this will reduce error in the model. In the example presented, this is not possible since there is no closed-form solution for the differential equation used in the dynamic blocks due to the $\partial x_i j/\partial t$ term.

Secondly, the parameter θ_i was fixed for each input. This was done because θ_i must be a multiple of 5 due to the fact that measurements were taken every five minutes. Since the objective function used here would not be continuous with respect to θ_i, one would be unable to calculate derivatives. Revising Algorithm 1 such that 5 is added or subtracted from $\widehat{\theta}_i$ would be one possible method of overcoming this. This could be done at the end of each stage of the algorithm as this would be done one $\widehat{\theta}_i$ at a time.

Another possible improvement is the elimination of constraints in the parameter space. Here the parameters τ_i and ζ_i must be larger than 0. One way to deal with this issue is reparameterization, which is suggested in [15]. By setting $\tau_i = e^{\lambda_i}$ and $\zeta_i = e^{\gamma_i}$, one can optimize with respect to λ_i and γ_i instead of τ_i and ζ_i. This would eliminate the need to check whether $\tau_i > 0$ or $\zeta_i > 0$ for any iteration of our methodology. The only concern is that approximating the Jacobian and the gradient will become even more difficult if this reparameterization is performed. One last thing to discuss is the importance of starting parameters. While finding starting dynamic parameter values that would yield useful models regardless of the data would make implementation easier, there may be properties of the experiment worth exploiting. It is common knowledge that carbohydrates are digested and metabolized faster than fats. Thus the amount of time that a step change in carbohydrates affects the system is less than that of fat. This "residence time" can be calculated for input i to be $2\tau_i\zeta_i$. For starting parameters, one idea could be to set the starting values for ζ_1 and ζ_2 to be equal and choose τ_1 to be less than τ_2.

Appendix

For these algorithms, let θ denote the vector of all parameters, $\widetilde{\theta}$ denote the subset of parameters to be fit, and $f(x|\theta)$ denote the model of interest, see Algorithms 2, 3, 4, and 5. Also note that checks for convergence have been left out. First we give a short legend of the notation used in this appendix.

$\widehat{\theta}$: Current estimate of parameters,

$\mathbf{f}$: $[f(\mathbf{x}_1 \mid \widehat{\theta})\ f(\mathbf{x}_2 \mid \widehat{\theta}) \cdots f(\mathbf{x}_n \mid \widehat{\theta})]$,

∇F: $\nabla F(\mathbf{x} \mid \widehat{\theta})$,

$F(\mathbf{x} \mid \widehat{\theta})$: $\sum (y_i - f(x_i \mid \widehat{\theta}))^2$,

J: Jacobian of $F(\mathbf{x} \mid \widehat{\theta})$.

```
Set $\alpha$.
Let $\nu = 2$
**For** $i = 1$ to 5 **do**
    Calculate the Jacobian $J$.
    **if** $i = 1$ **then**
        $\lambda = \max(J'J) \cdot \alpha$
    **else**
        $\lambda = \lambda \cdot \max(\frac{1}{3}, 1 - (2\eta - 1)^3)$
    **end if**
    $G = J'\mathbf{f}$
    $h = (J'J + \lambda I)^{-1}G$
    $\widehat{\theta} = \widehat{\theta} + h$
    $\eta = \frac{F(\widehat{\theta} - h) - F(\widehat{\theta})}{h'(\lambda h - G)}$
    **if** $\eta > 0$ **then**
        $\nu = 2$
    **else**
        $\lambda = \lambda \cdot \nu$
        $\nu = 2\nu$
        **if** $\nu > 128$ **then**
            break
        **end if**
    **end if**
**end for**
```

Algorithm 2: Levenberg-Marquardt algorithm used.

```
Let $n$ = number of iterations to be run and $\sigma = 1 \times 10^{-8}$.
**for** $i = 1$ to $n$ **do**
    **if** $i = 1$ **then**
        $h = -\nabla F$
    **else**
        $\beta = \max\left(0, \frac{\nabla F'(\nabla F - \nabla F(\mathbf{x} \mid \widehat{\theta} - h))}{\nabla F(\mathbf{x} \mid \widehat{\theta} - h)'\nabla F(\mathbf{x} \mid \widehat{\theta} - h)}\right)$
        $h = -\nabla F + \beta h$
    **end if**
    $k = -\nabla F'h$
    $\alpha = -\sigma \cdot k/(\nabla F(\mathbf{x} \mid \widehat{\theta} + \sigma h)'h - k)$
    $\widehat{\theta} = \widehat{\theta} + \alpha h$
**end for**
```

Algorithm 3: Conjugate Gradient algorithm used.

```
Choose $N$ = number of iterations to be run based on starting value of $R^2$ and $i = 1$.
Generate a simplex around the starting parameters $\hat{\theta}$; denote them $(\tilde{\theta}_1, \tilde{\theta}_2, \dots, \tilde{\theta}_{n+1})$.
While  $i \le N$
  Reorder the points of the simplex such that $F(\mathbf{x}|\tilde{\theta}_1) \le F(\mathbf{x}|\tilde{\theta}_2) \le \cdots \le F(\mathbf{x}|\tilde{\theta}_{n+1})$.
  $\overline{\theta} = \sum_{i=1}^{n} \tilde{\theta}_i$
  $\tilde{\theta}^* = 2\overline{\theta} - \tilde{\theta}_{n+1}$
  if $F(\mathbf{x} \mid \tilde{\theta}_1) \le F(\mathbf{x} \mid \tilde{\theta}^*) < F(\mathbf{x} \mid \tilde{\theta}_n)$ then
    $\tilde{\theta}_{n+1} = \tilde{\theta}^*$
    $i = i + 1$; next
  else if $F(\mathbf{x} \mid \tilde{\theta}^*) < F(\mathbf{x} \mid \tilde{\theta}_1)$then
    $\tilde{\theta}^{**} = 3\overline{\theta} - 2\tilde{\theta}_{n+1}$
    if $F(\mathbf{x} \mid \tilde{\theta}^{**}) < F(\mathbf{x} \mid \tilde{\theta}^*)$ then
      $\tilde{\theta}_{n+1} = \tilde{\theta}^{**}$
    else
      $\tilde{\theta}_{n+1} = \tilde{\theta}^*$
    end if
    $i = i + 1$; next
  else
    if $F(\mathbf{x} \mid \tilde{\theta}_n) \le F(\mathbf{x} \mid \tilde{\theta}^*) < F(\mathbf{x} \mid \tilde{\theta}_{n+1})$ then
      $\tilde{\theta}^{**} = \frac{3}{2}\overline{\theta} - \frac{1}{2}\tilde{\theta}_{n+1}$
      if $F(\mathbf{x} \mid \tilde{\theta}^{**} \le F(\mathbf{x} \mid \tilde{\theta}^*)$ then
        $\tilde{\theta}_{n+1} = \tilde{\theta}^{**}$
        $i = i + 1$; next
      end if
    else
      $\tilde{\theta}^{**} = \frac{1}{2}\overline{\theta} + \frac{1}{2}\tilde{\theta}_{n+1}$
      if $F(\mathbf{x} \mid \tilde{\theta}^{**} < F(\mathbf{x} \mid \tilde{\theta}_{n+1})$ then
        $\tilde{\theta}_{n+1} = \tilde{\theta}^{**}$
        $i = i + 1$; next
      end if
    end if
    for $i = 2, 3, \dots, n + 1$ do
      $\tilde{\theta}_i = \frac{1}{2}(\tilde{\theta}_1 + \tilde{\theta}_i)$
    end for
    $i = i + 1$
    end if
end while
```

Algorithm 4: Nelder-Mead algorithm used. (Taken from [6, 12]).

```
Let $n$ = number of iterations to be run, $\sigma = 1 \times 10^{-12}$, and $B^{-1} = I$.
Choose $\delta$ based on current value of $R^2$ and subset of parameters to be fit.
**for** $i = 1$ to $n$ **do**
    **if** $i > 1$ **then**
        $t = J'Jh + (J - J^*)'f$
        $B^{-1} = B^{-1} + \dfrac{h't + t'B^{-1}t}{t'hh't}hh' - \dfrac{B^{-1}th' + ht'B^{-1}}{h't}$
    **end if**
    $F^* = J'\mathbf{f}$
    $h = -B^{-1}F^*$
    $d = \sqrt{h'h}$
    **if** $d > \delta$ **then**
        $h = \dfrac{\delta}{d^2}h$
        $d = \delta$
    **end if**
    $\widehat{\theta} = \widehat{\theta} + h$
    $\rho = \dfrac{F(\mathbf{x} \mid \widehat{\theta} - h) - F(\mathbf{x} \mid \widehat{\theta})}{-h'F^* - (1/2)\|Jh\|^2}$
    **if** $\rho < .25$ **then**
        $\delta = \dfrac{\delta}{2}$
    **else**
        **if** $\rho > .75$ **then**
            $\delta = \max(\delta, 3d)$
        **end if**
    **end if**
        $J^* = J$
**end for**
```

Algorithm 5: BFGS algorithm used.

Acknowledgments

The authors would like to thank Jeanne Stewart and Kaylee Kotz for their help with data collection and BodyMedia, Inc. for their funding and donation of equipment.

References

[1] A. H. Tan and K. Godfrey, "Modeling of direction-dependent processes using Wiener models and neural networks with nonlinear output error structure," *IEEE Transactions on Instrumentation and Measurement*, vol. 53, no. 3, pp. 744–753, 2004.

[2] D. K. Rollins, N. Bhandari, J. Kleinedler et al., "Free-living inferential modeling of blood glucose level using only noninvasive inputs," *Journal of Process Control*, vol. 20, no. 1, pp. 95–107, 2010.

[3] G. Pajunen, "Adaptive control of Wiener type nonlinear systems," *Automatica*, vol. 28, no. 4, pp. 781–785, 1992.

[4] J. D. Faires and R. Burden, *Numerical Methods*, Brooks/Cole, Pacific Grove, Calif, USA, 2nd edition, 1998.

[5] T. Hastie, R. Tibshirani, and J. Friedman, *The Elements of Statistical Learning*, Springer Series in Statistics, Springer, New York, NY, USA, 2nd edition, 2009.

[6] *MATLAB Version 7.10.0.*, The MathWorks, Natick, Mass, USA, 201.

[7] K. Madsen, H. B. Nielsen, and O. Tingleff, *Methods for Non-Linear Least Squares Problems*, Informatics and Mathematical Modelling, Technical University of Denmark, Kongens Lyngby, Denmark, 2nd edition, 2004.

[8] R Development Core Team, *R: A Language and Environment for Statistical Computing*, R Foundation for Statistical Computing, Vienna, Austria, 2010.

[9] R. H. Barham and W. Drane, "An algorithm for least squares estimation of nonlinear parameters when some of the parameters are linear," *Technometrics*, vol. 14, pp. 757–766, 1972.

[10] H. O. Hartley, "The modified Gauss-Newton method for the fitting of non-linear regression functions by least squares," *Technometrics*, vol. 3, pp. 269–280, 1961.

[11] L. S. Lasdon and A. D. Waren, "Generalized reduced gradient software for linearly and nonlinearly constrained problems," in *Design and Implementation of Optimization Software*, H. J. Greenberg, Ed., pp. 335–362, Sijthoff and Noordhoff, Groningen, The Netherlands, 1978.

[12] J. Nocedal and S. J. Wright, *Numerical Optimization*, Springer Series in Operations Research, Springer, New York, NY, USA, 1999.

[13] J. R. Shewchuk, "An Introduction to the Conjugate Gradient Method Without the Agonizing Pain Edition 1 1/4," 1994, http://www.cs.cmu.edu/~quake-papers/painless-conjugate-gradient.pdf.

[14] E. Van Cauter, K. S. Polonsky, and A. J. Scheen, "Roles of circadian rhythmicity and sleep in human glucose regulation," *Endocrine Reviews*, vol. 18, no. 5, pp. 716–738, 1997.

[15] D. M. Bates and D. G. Watts, *Nonlinear Regression Analysis and Its Applications*, Wiley Series in Probability and Mathematical Statistics: Applied Probability and Statistics, John Wiley & Sons, New York, NY, USA, 1988.