Final Year Project II

Final Draft

Estimation of GOR at reservoir pressures Below bubble point pressure using GMDH (Group Method of Data Handling)

Moctar Bebaha
(14209)

Supervised by: Dr. Mohammed Abdalla Ayoub

May 2014

Table of Content:

CERTIFICATION OF APPROVAL

Estimation of Solution Gas Oil Ratio at pressures below Bubble Point Pressure using Group Method of Data Handling

by

Moctar Bebaha

14209

A project dissertation submitted to the

Petroleum Engineering Programme

Universiti Teknologi PETRONAS

In partial fulfilment of the requirement for the

BACHELOR OF ENGINEERING (Hons)

(PETROLEUM)

Approved by,

_____

Dr. Mohammed Abdalla Ayoub Mohammed

CERTIFICATION OF ORIGINALITY

This is to certify that I am responsible for the work submitted in this project, that the original work is my own except as specified in the references and acknowledgements, and that the original work contained herein have not been undertaken or done by unspecified sources or persons.

_____

Moctar Bebaha

List of Figures:

I- Abstract:

A GMDH neural networks modelling approach is proposed to estimate gas oil ratio at pressures below bubble point pressure. A new correlation is developed by the use of 385 PVT data collected from available literature on GOR measurement and ranging from 100scf/STB to a little over 1500scf/STB. GMDH approach is explained and an overview of the available literature on GMDH modelling is laid out. The new correlation is multinomial algebraic in nature, and is tested against the currently widely used correlations in the petroleum industry. Correlations factors for all correlations are produced.

II- Introduction:

-Solution Gas Oil Ratio:

Solution Gas Oil Ratio (GOR) or R_s is a measure or indication of the quantity of hydrocarbon gas that is dissolved in reservoir liquids at reservoir pressures (Ahmed, 2007). GOR is defined as the ratio of the volume of gas that comes from the oil produced (or water) at the surface (atmospheric pressure) measured in standard cubic feet (scf) to the volume of oil produced after the dissolved gas has evolved from it at the surface, measured in stb.

$$R_{so} = V_{gas}/V_{oil}$$

V_{gas}: volume of gas that has evolved from the oil produced at the surface

V_{oil}: Volume of oil produced after the gas evolved from it at atmospheric pressure

The solution gas oil ratio R_s or GOR varies usually in the range of 0 scf/stb to 2100 scf per STB. 0 scf per STB is found in dead oil that has no gas at all. While 2100 scf per STB is encountered with very light oil or gas condensate.

R_s is usually plotted against pressure, and the plot is characterized by a constant value before the bubble point pressure is reached (from high pressure) until the bubble point pressure of oil P_b is reached, after which the GOR decreases almost linearly to 0. (Archer, 1986)

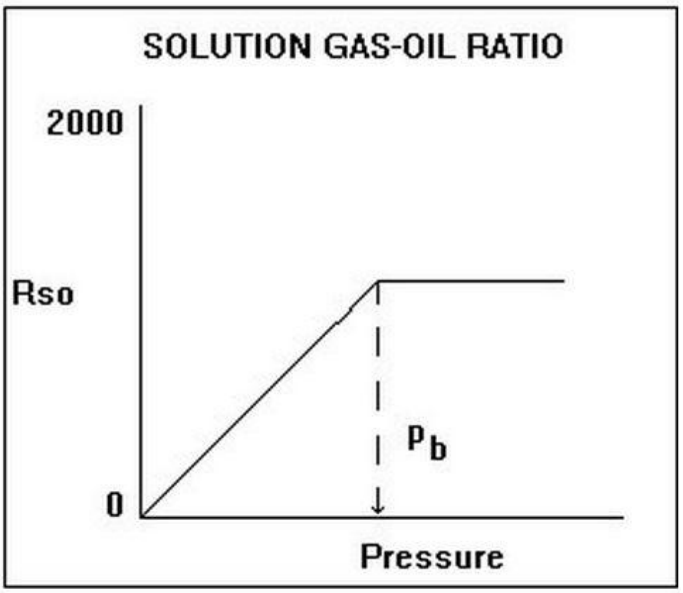


Fig. 1. Gas Oil Ratio against Pressure

Knowing gas oil ratio is important for production and reservoir engineers since it has a great influence on reservoir fluid properties such as formation volume factor and viscosity. Therefore, in the past few decades, many correlations were devised and developed to estimate gas oil ratio for the pupose production optimization and formation evaluation (Franchi, 2006).

a) Background:

Standing, in 1947, published what went to become one of the most famous correlations in the oil and gas industry. He was the first to undertake an extensive PVT analysis on a number data gathered from oilfields attempting to develop a correlation for Bubble Point Pressure Pb and Oil Formation Volume Factor Bo.

The correlation was of great importance due to the necessity of estimating Solution Gas Oil Ratio, due to the lack of actual pvt data most of the time, and to the difficulty encountered often when attempting to predict reservoir fluid properties throughout the oil field lifespan. This is mostly due to the number of variables that have to be taken into consideration and to the difficulty and the expenses associated with running lab pvt tests. It is not surprising therefore that many correlations have followed Standing's, and were developed estimate bubble point pressure and other fluid properties.

Standing's correlation is:

$$\mathbf{P_b = 18.2*((R_{si}/\gamma_g)^{0.83}(10^{0.00091T-0.0125\gamma_{API}})-1.40),}$$ and

$$\mathbf{B_o = 12*10^{-5}*(R_s(\gamma_g / \gamma_o)^{0.5} + 1.5*T)^{1.2} + 0.9760}$$

P_b : Bubble point pressure

R_{si} : Initial gas oil ratio

R_s : gas oil ratio

γ_g :Gas specific gravity

γ_o :Stock tank oil gravity

γ_{API} API specific Gravity

T: Tempetrature in Fahrenheit

Numerous correlations have been developed since Standing's correlations were published in 1947. These generally provided wider range of applicability and more accurate estimations of the various fluid properties. The boost came from fast computers and the ever increasing need in the petroleum industry for more accurate estimations. (Galso,1980), (Al-Marhoun,1988), (Farshad ,1990), (Valko,2003) and (Veralde,1997) are some of the widely used correlations developed for different regions and different types of Oil.

Sing et al (2012) utilized field information from Trinidad Oil fields to assess the aforementioned correlations on liquid properties, for example, formation volume factor and viscosity, bubble point pressure and gas oil ratio with the end goal of store estimation.

Veralde et al (1997) created a set of correlations that is pertinent on more extensive extent of temperatures, gas oil ratio. It was found that their correlations gave the most correct estimations regarding air bubble point pressure, gravity and other fluid properties in case of Trinidad oil fields.

The average absolute deviation AAD on account of Veralde et al correlations was short of what 5%, while the deviation for different correlations was much higher (Sing et al, 2012).

b) Problem Statement:

Although many gas oil ratio correlations have been developed in the past 50 years, the vast majority of them have limited use on a limited range of values if they are to be of any accuracy. As pointed out by Veralde (1997), in case of the Gas oil ratio, most correlations yield significant errors when applied at pressures below bubble point pressure and with high initial gas oil ratio. The typical concave up-point of inflection-concave down shape that is characteristic of GOR curves vs. pressure is seldom captured in GOR correlations.

Fig. 2 shows typical deviations from actual lab and field data encountered when using most of the available correlations to estimate Gas Oil Ratio. These deviations are their maximum when the initial GOR is very high.

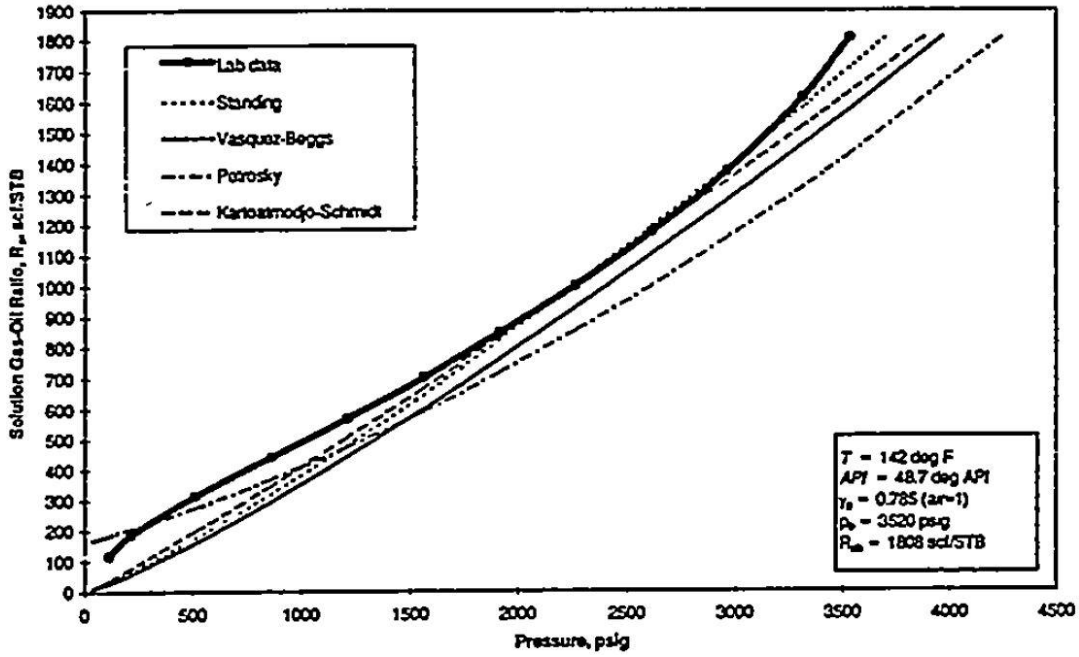


Fig. 2. A Comparison of gas oil ratio prediction by several correlations with actual Laboratory Data Veralde, (1997)

In terms of Average absolute deviation AAD the deviations can be in the range 20-30% of (Sing, 2012).

Therefore, a new set of correlations should be developed, one that takes into consideration the fact that the curve of a typical curve of gas oil ratio vs. Pressure, is concave up inflexed and concave down at high initial gas oil ratio, and which can cover a wider range of GOR values.

c) Objectives and Scope of study:

The aims and objectives of this paper are to create another correlation for gas oil ratio GOR, more accurate than other correlations, at reservoir pressures below bubble point pressure. Group Method of Data Handling (GMDH), a method of Artificial Neural Networks ANN is to be utilized. The usage of Neural Network model dependent upon GMDH has been chose due to the popularity of this field of study and the encouraging results from its application in similar areas more recently. The

model will be prepared on various sets of PVT data to produce suitable working model which then might be connected to new data and produce correct effects.

III- Group Method of Data Handling:

Let y, x_1 x_2 …. and x_m be:

y : Solution gas oil ratio

x_1 : Bubble point pressure

x_2 : Gas specific gravity

x_3

.

.

.

x_m = API specific Gravity

x_1 to x_m are variables upon which solution gas oil ratio depends.

The purpose of a GMDH model is to find constants a_i, a_{ij}, and a_{ijk} such that:

$$y = a_0 + \sum_{i=1}^{m} a_i x_i + \sum_{i=1}^{m}\sum_{j=1}^{m} a_{ij} x_i x_j + \sum_{i=1}^{m}\sum_{j=1}^{m}\sum_{k=1}^{m} a_{ijk} x_i x_j x_k$$

This equation is in fact a correlation of y (solution gas oil ratio) to x_1 to x_m (variables such as specific gravity, formation volume factor etc.) upon which it depends.

The key to the success of the model is the estimation of the factors a_i called weights. The method consists of a combination of polynomial regressions combined in a

network and trained on a set of data in the input and output to extract the suitable weights. This process is theoretically simple, but requires a lot of computation that can only be done with mathematical software such as Matlab.

This is a sample of the data used to develop the correlation in this research. From Gassan (1988)

RS	**P**	**API**	**GG**	**T**	**GO**
92.7	563.0	26	0.95	212	0.898413
51.0	512.0	14.7	0.72	123.008	0.967852
877.5	3642.0	32.8	0.812	244.04	0.861229
334.3	1740.0	31	0.7	170.96	0.870769
742.5	3200.0	32	0.75	170.06	0.865443
153.1	1200.0	28	0.62	145.004	0.887147
240.4	1422.0	32	0.75	170.06	0.865443
168.5	600.0	26	0.94	140	0.898413
573.0	2296.0	32.6	0.756	186.98	0.862279
361.2	1462.0	36.3	0.816	136.4	0.843266
76.1	666.0	12	0.73	116.06	0.986063
509.0	2417.6	24	0.75	170.06	0.909968
406.2	1728.0	37.4	0.79	150.08	0.837774
284.8	1720.0	32	0.75	170.06	0.865443
57.0	555.0	12	0.68	105.98	0.986063
660.7	3394.9	35.7	0.681	242.6	0.846292
782.6	2885.9	38.4	0.713	150.08	0.832843

321.3	1409.0	49.1	0.673	179.96	0.783499
736.5	3683.9	20.9	0.75	170.06	0.928478
357.3	1200.0	30	0.95	120.02	0.876161
1007.3	3053.0	41.4	0.708	114.98	0.818392
679.8	1846.0	53.4	0.792	224.96	0.765279
429.3	2133.1	23	0.75	177.8	0.915858
372.1	1400.0	34	0.8	100.04	0.854985
316.3	1505.6	33	0.75	170.06	0.860182

Table 3: PVT data Middle East field (Ghassan, 1988)

The remaining data points can be found in the appendix I.

IV- Literature Review:

Artificial Neural Networks have seen an expanding enthusiasm since their initiation by in the 1940s. They have been utilized within data mining in diverse fields, from Medical Sciences to Economy, Engineering and Earth Sciences (Maryam, 2013).

In the field of Petroleum Engineering, neural networks have been utilized within the assessment of underneath bubble point consistency connections (Ayoub et al 2007). A neural system model was created to foresee the thickness underneath the bubble point pressure with a correspondence coefficient of 99.3%. Ayoub et al (2005) likewise created a neural system model for anticipating bottomhole streaming pressure in vertical multiphase stream with a correspondence coefficient of 97.35%.

Gas Oil Ratio Rs has additionally been subject to neural networks models. Sarit Dutta and J.p. Gupta (2008) created neural models for Oil Formation Volume Factor,

Bubble Point Pressure, Viscosity and Gas Oil Ratio, for Indian West Coast Crude Oil. Their model, they claimed, performed better than most existing connections by giving much lower normal total relative error. Asadisaghandi et al (2009) assessed back spread neural system taking in calculations and exact correspondences for forecast of PVT properties in Iran Oilfields.

One normal rule around all distributed writing on neural networks demonstrates in oil and gas industry is the affirmation that the correspondences created by the neural models are exceedingly faultless beat very nearly all existing relationships. They likewise assert high adaptability in the model and the way that no real understanding of the internal laws of the framework at study is required. Actually one property the neural system demonstrate being referred to is that they are verifiable models, implying that they can display complex frameworks gave a set of inputs and the relating alluring yields are accessible, without the need for the learning of the inward working of the framework (Ivakhnenko, 1968).

V- Methodology:

Lab and Field Data has been assembled (see appendix _and gathered from accessible writing and PVT Databanks on the web and Library Journals. The data concerns the various inputs and the single output that are to be demonstrated (Pb Bubble point pressure, Gg Gas specific gravity, Go Stock tank oil gravity, Grapi API specific Gravity and Temperature) and (Rs) individually. A GMDH neural system model will be created and a suitable back proliferation calculation will be connected to it, to recognize the fitting synaptic weights. Matlab software will be utilized as a part of the scientific estimations to help process the synaptic weights and evaluate the last manifestation of the relationship.

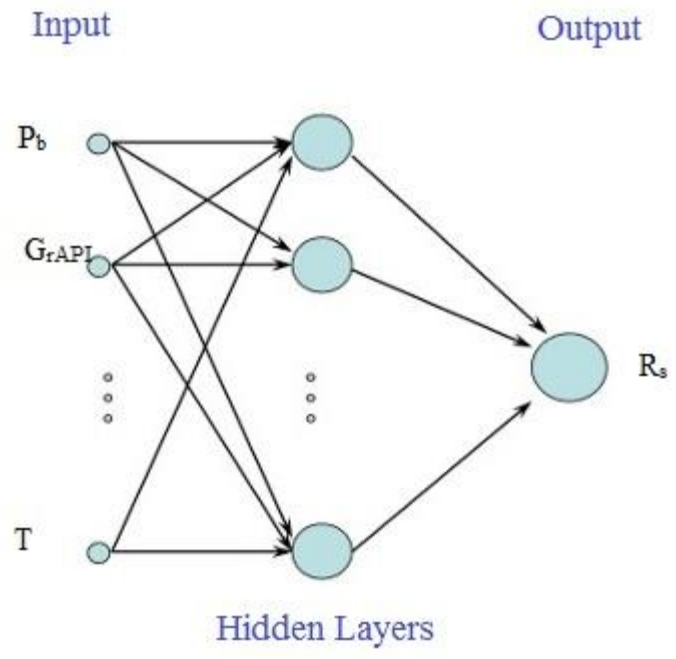
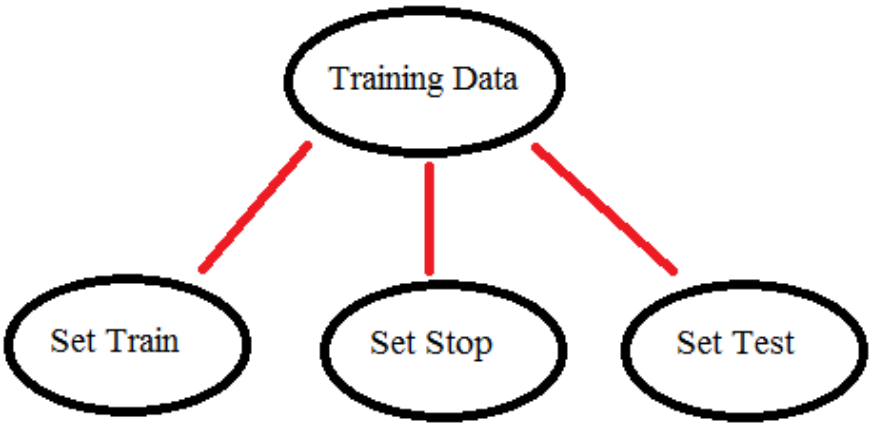


Fig. 3. The GMDH network with multiple inputs and a single output

Two real issues that need to be alleviated and managed and that are often experienced when planning such neural system models are the issue of Generalization and the issue of Over-fitting. Generalization alludes to the inquiry whether the model, which has been prepared to a certain set of data, could be summed up (connected) effectively to other set of data that are not like the preparation dat and produce correct yield. Over-fitting alludes to the situation when the system endeavors to model all the data, including the clamor and superfluous data, as opposed to discovering the relationship that underlies it.

As a way of mitigating these two problems and ensuring they don't complicated the model, we have designed our training data in such a way that it's divided into three sets of data which we have termed Train,Stop and Test.



Set Train will be the set of data the system is to be prepared with. Commonly, as we prepare the system, its execution ought to expand on an autonomous set. That set will be Set Stop. The reason for Set Stop is to check for over-fitting. As long as system is advancing in the right course, the execution on the Set Stop ought to continue expanding. On the other hand, there will be a point when the system begins to once again fit, by then, its execution on Set Stop ought to go down. That is the point at which the Network preparing ought to stop. Set Test will be utilized for the assessment of the genuine execution of the system when the procedure is carried out. The execution on the Set Test will be viewed as the true execution of the Neural Network GMDH Model correlation.

The data used in creating this correlation is available in the published literature. 385 PVT data points consisting of 5 reservoir fluids parameters (Pressure (kPa), Temperature(C), API, Gas specific gravity and Gas Oil Ratio (m3/m3)) were chosen from Ghassan (1988). This data has not been used prior to its publication in creating any correlation. The data was subsequently converted into field data for more suitable treatment.

The common practice of dividing the data into three sets was followed. The choice was made, after randomizing it, to divide it into 50%, 25% and 25% also known as (2:1:1) partitioning. 50% was assigned for training, 25% for validation, and the remaining 25% for testing.

Matlab was then used to produce the suitable network architecture. The code used is a modified version of the original code created by Ayoub (2011) in his estimation of pressure drop in pipelines by abductive (GMDH) neural networks. The changes

made to the original code included the type of input data, the number of data point, the number of independent input variables and the number of layers etc.

VI- Results:

A Four-Layer model was produced with, as pre-defined, 4 independent parameters (Pressure, Temperature, Gas Gravity and API) and one dependent parameter, Gas Oil Ratio Rs. The output of the program was as follows:
Number of layers: 4
Layer #1

x5 = 40 -0.0816*x4 -3.15*x2 +0.0794*x1 -0.0241*x2*x4 -0.00105*x1*x4 +0.0107*x1*x2 +0.00452*x4*x4 +0.025*x2*x2 +6.15e-006*x1*x1

Layer #2

x9 = -418 +0.0976*x5 +922*x3 +0.0245*x1 +0.505*x3*x5 -0.000201*x1*x5 +0.0943*x1*x3 +0.000649*x5*x5 -503*x3*x3 +1.88e-005*x1*x1

Layer #3

x13 = -79.4 +0.679*x9 +1.77*x4 -70*x3 +0.000994*x4*x9 +0.178*x3*x9 -0.587*x3*x4 +2.03e-005*x9*x9 -0.00497*x4*x4 +100*x3*x3

Layer #4

y = -38.2 +0.782*x13 +0.438*x2 +0.0754*x1 +0.00519*x2*x13 -6.8e-006*x1*x13 -0.00228*x1*x2 +2.7e-005*x13*x13 +0.0343*x2*x2 +5.55e-007*x1*x1

x1= Pressure in psi P
x2= API Gravity API
x3= Gas Gravity GG and,
x4= Temperature in F T
y= Gas Oil Ratio Rs

x5, x9 and x13 are dependent variables produced by the models in is internal layers that are dependent upon the input variables. We shall call them A, B and M respectively.

The correlation therefore becomes:

Rs = -38.2 +0.782*M +0.438*API +0.0754*P +0.00519*API*M -6.8e-006*P*M -0.00228*P*API +2.7e-005*M^2 +0.0343*API^2 +5.55e-007*P^2

A = 40 -0.0816*T -3.15*API +0.0794*P -0.0241*API*T -0.00105*P*T +0.0107*P*API +0.00452*T^2 +0.025*API^2 +6.15e-006*P^2

B = -418 +0.0976*A +922*GG+0.0245*P +0.505*GG*A -0.000201*P*A +0.0943*P*GG +0.000649*A^2 -503*GG^2 +1.88e-005*(P)^2

M = -79.4 +0.679*B+1.77*T -70*GG +0.000994*T*B +0.178*GG*B -0.587*GG*T +2.03e-005*B^2 -0.00497*T^2 +100*GG^2

The training stopped when the correlation achieved a 99% correlation factor R on the training set. The corresponding correlation factor on the validation set stood at 99.23%, while the actual correlation factor of the model on the independ testing set stood at 98.63%.

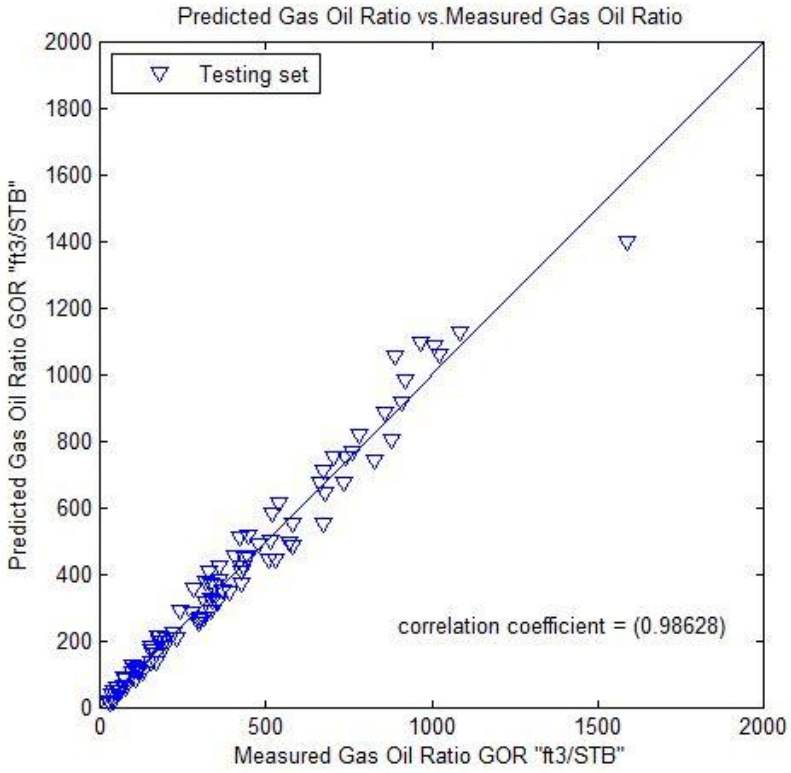


Fig. 2- X-Y Plot of Measured vs. Predicted Rs with the correlation coefficient

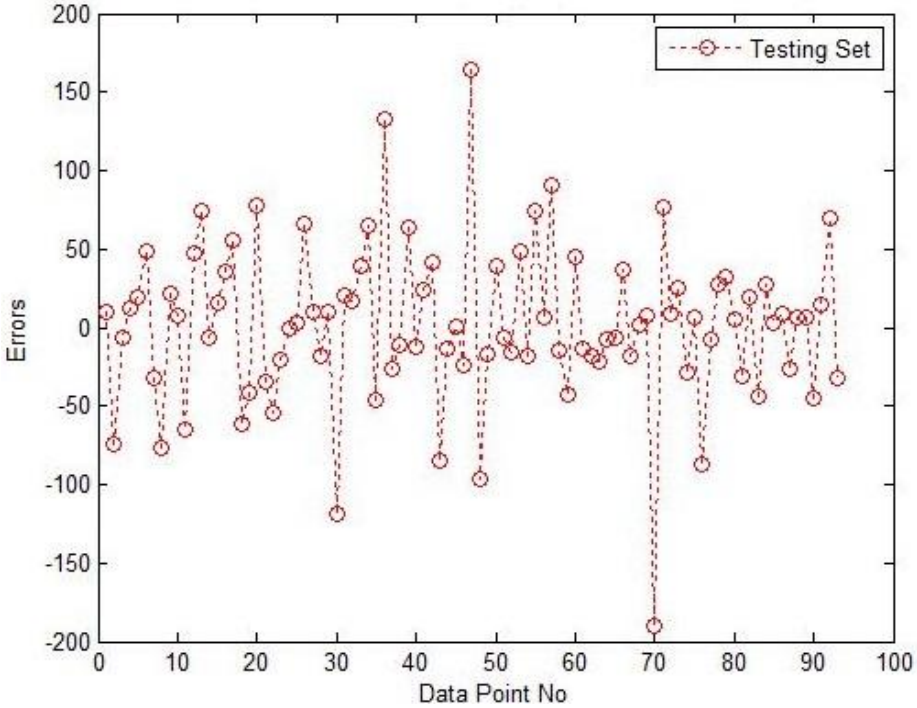


Fig. 3- Residual Graph for the Testing Set (Errors vs. Testing Points)

VII- Discussion and Comparison:

Three statistical parameters are going to be used to compare the performance of the new correlation relative to avalaible correlations.

- The mean absolute percentage error defined as:

  (Σ(actual value – predicted value)/actual value * n)*100

- The mean absolute deviation MAD defined as:

  (Σ(actual value – predicted value)/n)*100

And,

- Correlation coefficient

Six widely used correlations were applied to the testing data and their Mean Absolute Percentage and Deviation Errors, as well as their correlation factors were computed. The correlations are: Standing's, Vasquez and Beggs, Al-Marhoun, Lasater, Galso, and Petrosky correlations.

The results are as follows:

MAPE: Mean Absolute Percentage Error

MAD: Mean Absolude Deviation

R: Correlation Coefficient

	MAPE %	MAD (scf/STB)	R %
Standing	13.7	43.7	87.7
Al-Marhoun	23.5	71.53	83.1
Lasater	27.23	54.66	85.9
Vasquez and Beggs	13.25	44.42	86.96
Galso	21.24	65.88	88.74
Petrosky and Farshad	41.2	66.1	85
Ayoub & Moctar	11.93	35.68	98.63

Table 1- Comparison of the new correlation to the established correlations

The new correlation has the lowest absolute percentage error at 11.93%. Only Standing's and Vaquez correlations come close, at 13.7% and 13.25% respectively. In terms of absolute deviation, measuring the deviation of the predicted value from the actual value, the new correlation outperforms the existing tested correlations, with an average of 35.68scf/STB. Again, Standing and Vasquez are the closest to matching it, with 43.7 and 44.42scf/STB.

In terms of correlation coefficient, none of the widely used correlations tested here achieved abot 89%. Our correlation has an impressive 98.63 correlation coefficient.

VIII- Conclusion:

A new correlation for Gas Oil Ratio at pressures below bubble point pressure has been developed using Group Method of Data Handling. The new correlation has a correlation coefficient of 98%. It has an average absolute deviation of less than 36scf/STB and an average percentage error of less than 12%.

The correlation has been compared to the most widely used correlations in the industry and has outperformed them in terms of correlation coefficient, absolute and average percentage error. The new correlation underlines the importance and the potential of artificial neural networks in the field of petroleum engineering.

The use of Artificial Neural Networks and in particular GMDH algorithms (Group Method of Data Handling GMDH) to model Solution Gas Oil Ratio at reservoir pressures below bubble point pressure appears to be promising. More dedicated research and development need to be carried out for more accurated and generalized models to be developed. Future correlations need to be simpler in form and need to have fewer paramters.

References:

Ahmed, T. (2007). *Equations of States and PVT Analysis*. Huston, Texas: Gulf Publishing Company.

Archer, J. S., & Wall, C. J. (1986). *Petroleum engineering, principles and practices*. London, UK: British Library Cataloguing in Publication data.

Ayoub. M.A, SPE, Raja, D.M, Al-Marhoun, M.A. (2007). Evaluation of below Bubble Point viscosity correlations & construction of a new Neural Network Model. *SPE 108439*

Ayoub. M.A, Osman El-Sayed, A. (2005) An Artificial Neural Network Model for Predicting Bottomhole Flowing Pressure in Vertical Multiphase Flow. 93632-MS SPE

Franchi, J., R. (2006). *Principles of Applied Reservoir Simulation.* Burlington, MA: Gulf Professional Publishing.

Ghassan Abdul-Majeed, Naeema H. Salman (1988). Statistical Evaluations of PVT correlations- Solution Gas Oil Ratio. JCPT Reservoir Engineering

Ivakhnenko, A.G. (1968). The Group Method of Data Handling-A rival of the

Method of Stochastic Approximation. *Soviet Automatic Control*, 13 (3), 43-55.

Jalil Asadisaghandi, Pejman Tahmasebi (2011). Comparative evaluation of back-propagation neural network learning algorithms and empirical correlations for prediction of oil PVT properties in Iran oilfields. *Journal of Petroleum Science and Engineering. 78 (2011) 464–475*

Sarit Dutta, J.P. Gupta (2010). PVT Correlations for Indian Crude using Artificial Neural Networks. Journal of Petroleum Science and Engineering, 72, 93–109

APPENDIX I:

Other data that will be included in further testing and training of the final correlation from Al-Marhoun (1988), and Dokla and Osman (1990)

Rs (scf/STB)	ρ_{ob} (lb/ft^3)	P_b (psia)	B_{ob} (bbl/STB)	γg (air=1)	API	T F
1011	37.840	2090	1.68	1.05	48.2	210
494	41.610	2081	1.315	0.677	44.50	230
765	38.650	2058	1.52	0.939	48.80	205
491	44.470	2020	1.321	1.051	39.20	211
415	47.450	1982	1.246	1.14	36.10	224
367	45.020	1951	1.23	0.627	37.50	173
384	46.550	1910	1.238	0.733	32.60	152
366	46.690	1838	1.208	0.664	34.80	153
285	50.800	1818	1.153	0.704	26.60	152
606	38.550	1810	1.423	0.77	50.50	189
599	38.910	1805	1.424	0.767	48.10	204
686	38.040	1790	1.496	0.8	47.10	224
509	42.630	1780	1.362	0.853	37.80	205
585	39.240	1769	1.401	0.765	49.10	204
345	47.810	1765	1.184	0.695	34	151
372	49.410	1760	1.222	1.195	31	211
628	38.550	1758	1.442	0.762	48.4	199
694	38.000	1755	1.48	0.79	49.50	190
714	37.970	1750	1.5	0.82	48.70	189
524	42.650	1744	1.325	0.727	40.50	190
563	38.960	1741	1.409	0.759	48.40	217
397	44.500	1728	1.259	0.941	41.80	215
364	46.770	1700	1.232	1.028	36.60	206
646	42.580	1698	1.408	0.964	40	193
421	48.980	1660	1.221	1.298	37.10	203
368	45.710	1658	1.212	0.865	41.4	186
404	43.700	1620	1.265	0.847	42.9	188
421	45.980	1593	1.268	1.181	39.80	203
366.0	47.000	1570	1.241	1.315	39	207
463	47.490	1562	1.261	1.281	38.9	196
355	47.550	1530	1.24	1.228	35	209
566	42.170	1530	1.334	0.817	45.2	185
522	39.870	1510	1.365	0.73	47.8	189
341	46.290	1492	1.201	0.716	37.4	159

359	48.610	1450	1.214	1.25	35.4	208
425	46.330	1414	1.249	1.155	41	185
287	48.830	1390	1.154	0.718	33.4	141
313	47.840	1370	1.192	1.174	38.2	205
242	48.640	1302	1.17	0.824	31.4	180
198	50.070	1271	1.139	0.775	29.2	187
267	48.200	1225	1.176	1.263	38	211
260	48.050	1225	1.17	1.168	38	211
267	48.950	1220	1.173	0.884	31.4	174
214	48.580	1195	1.152	0.664	31.9	180
169	50.040	1085	1.128	0.638	29.1	187
220	49.800	1058	1.13	0.79	32.3	127
142	52.230	952	1.092	0.667	26.9	146
274	47.340	790	1.168	1.005	39.8	150
1507	37.970	3573	1.875	0.951	39.3	225
898	43.210	3571	1.471	0.802	32.7	175
898	43.810	3426	1.451	0.802	32.7	150
1579	35.370	3405	1.997	0.93	42.8	235
825	43.350	3354	1.431	0.779	34.2	185
825	43.530	3311	1.425	0.779	34.2	175
867	42.750	3297	1.458	0.799	35.4	180
898	44.450	3279	1.43	0.802	32.7	125
1203	38.10	3250	1.747	0.925	40.2	240
775	43.510	3228	1.413	0.783	34.4	175
750	44.820	3223	1.387	0.8	32	175
1151	38.850	3218	1.686	0.894	39.9	110
742	44.750	3204	1.372	0.752	32.6	160
1579	36.790	3201	1.92	0.93	42.8	190
1602	35.780	3198	1.986	0.96	44.6	230
730	43.940	3160	1.392	0.757	33.1	175
700	44.30	3155	1.384	0.774	32.2	185
818	43.490	3155	1.427	0.789	34.2	170
898	45.050	3127	1.411	0.802	32.7	100
700	44.550	3101	1.376	0.774	32.2	175
680	45.410	3090	1.36	0.755	29.7	175
867	43.890	3066	1.42	0.799	35.4	140
811	42.570	3057	1.445	0.812	36.5	185
679	44.630	3057	1.371	0.778	32	175
1151	40.040	3030	1.636	0.894	39.9	180
665	45.770	3003	1.34	0.766	30.8	175

811	43.290	2941	1.421	0.812	36.5	160
693	43.320	2925	1.406	0.774	33.2	175
700	45.340	2901	1.352	0.774	32.2	140
818	45.470	2900	1.365	0.789	34.2	100
1579	38.140	2896	1.852	0.93	42.8	145
825	45.340	2871	1.368	0.779	34.2	100
742	46.270	2865	1.327	0.752	32.6	100
1143	39.510	2845	1.682	0.951	39.4	240
811	43.840	2836	1.403	0.812	36.5	140
1203	40.530	2831	1.642	0.925	40.2	160
867	45.030	2804	1.384	0.799	35.4	100
775	45.470	2789	1.352	0.783	34.4	100
750	46.630	2751	1.333	0.8	32	100
680	47.360	2687	1.304	0.755	29.7	100
1507	41.440	2652	1.718	0.951	39.3	100
700	46.340	2639	1.323	0.774	32.2	100
1143	40.350	2636	1.647	0.951	39.4	200
811	44.870	2617	1.371	0.812	36.5	100
679	46.530	2607	1.315	0.778	32	100
665	47.770	2588	1.284	0.766	30.8	100
1579	39.550	2559	1.786	0.93	42.8	100
602	45.540	2558	1.323	0.803	33	170
693	45.150	2530	1.349	0.774	33.2	100
746	42.980	2521	1.44	0.907	36.1	200
1151	42.320	2504	1.548	0.894	39.9	100
585	45.190	2445	1.329	0.815	33.3	180
1203	42.230	2413	1.576	0.925	40.2	100
567	44.930	2401	1.318	0.782	34.5	175
805	41.870	2392	1.479	0.929	39.1	200
498	46.950	2365	1.279	0.798	30.1	175
521	47.340	2359	1.274	0.801	30.1	160
1602	39.720	2350	1.789	0.96	44.6	100
1143	41.560	2344	1.599	0.951	39.4	150
521	47.980	2259	1.257	0.801	30.1	135
585	46.20	2256	1.3	0.815	33.3	140
469	47.440	2249	1.272	0.824	28.8	165
746	44.270	2231	1.398	0.907	36.1	150

580	44.370	2230	1.316	0.802	38.1	175
421	51.220	2177	1.213	0.799	21.9	145
602	47.330	2172	1.273	0.803	33	100
1493	40.880	2172	1.734	1.008	43.6	100
585	46.70	2148	1.286	0.815	33.3	120
805	43.250	2133	1.432	0.929	39.1	150
521	48.640	2132	1.24	0.801	30.1	110
692	42.080	2124	1.406	0.876	41.9	185
585	47.220	2035	1.272	0.815	33.3	100
803	43.880	2016	1.452	1.013	36.2	160
521	49.360	1990	1.222	0.801	30.1	85
692	43.030	1988	1.375	0.876	41.9	150
498	48.970	1981	1.226	0.798	30.1	100
746	45.710	1962	1.354	0.907	36.1	100
469	49.130	1928	1.228	0.824	28.8	100
585	47.780	1912	1.257	0.815	33.3	80
580	46.380	1890	1.259	0.802	38.1	100
805	44.650	1847	1.387	0.929	39.1	100
755	43.510	1834	1.425	1.004	39.3	170
692	44.020	1824	1.344	0.876	41.9	115
1087	44.160	1766	1.533	1.056	38	100
692	45.060	1641	1.313	0.876	41.9	80
803	45.610	1631	1.397	1.013	36.2	100
347	50.230	1630	1.203	0.933	26.1	165
755	44.70	1603	1.387	1.004	39.3	125
412	46.710	1480	1.28	0.973	31	180
560	44.870	1477	1.327	1.002	38.6	150
417	47.220	1472	1.267	0.98	31.2	185
389	49.420	1437	1.226	1.002	28.2	150
347	51.870	1405	1.165	0.933	26.1	100
412	47.490	1405	1.259	0.973	31	160
417	47.860	1378	1.25	0.98	31.2	160
331	49.060	1377	1.21	0.921	28.4	160
755	46.030	1367	1.347	1.004	39.3	80
412	48.290	1292	1.238	0.973	31	130
469	45.450	1282	1.291	0.96	36.5	155
417	48.680	1265	1.229	0.98	31.2	130

302	49.550	1230	1.188	0.931	28.9	160
389	51.480	1205	1.177	1.002	28.2	80
469	47.10	1193	1.246	0.96	36.5	130
412	49.170	1180	1.216	0.973	31	100
331	51.350	1180	1.156	0.921	28.4	100
512	47.10	1159	1.262	1.01	37	100
417	49.530	1153	1.208	0.98	31.2	100
433	48.320	1095	1.268	1.188	31.2	190
265	51.730	1094	1.18	1.058	22.8	185
302	51.10	1061	1.152	0.931	28.9	100
433	49.210	966	1.245	1.188	31.2	150
232	51.0	874	1.152	0.989	27.2	160
196	49.50	854	1.141	0.942	32.1	175
265	53.920	847	1.132	1.058	22.8	100
433	50.420	804	1.215	1.188	31.2	100
189	52.670	697	1.102	1.031	27.9	80
196	51.490	696	1.097	0.942	32.1	100
266	46.410	642	1.22	1.192	37.3	165
266	47.540	601	1.191	1.192	37.3	145
127	52.200	584	1.114	1.025	25.1	160
45	54.020	263	1.079	1.123	21.8	190
44	50.530	261	1.093	1.05	30.2	205
61	52.530	255	1.086	1.272	26.2	160
45	54.730	246	1.065	1.123	21.8	160
61	53.510	240	1.066	1.272	26.2	140
44	51.520	238	1.072	1.05	30.2	165
61	52.660	236	1.09	1.356	25.4	190
80	51.880	236	1.091	1.297	28.5	155
45	55.460	231	1.051	1.123	21.8	130
61	54.480	214	1.047	1.272	26.2	100
44	52.50	214	1.052	1.05	30.2	125
61	53.40	211	1.075	1.356	25.4	160
39	55.770	205	1.061	1.251	19.4	160
61	54.20	186	1.059	1.356	25.4	130
29	53.390	186	1.075	1.185	23.6	190

39	56.630	179	1.045	1.251	19.4	120
29	54.10	174	1.061	1.185	23.6	160
46	50.540	174	1.039	1.105	38.9	100
26	51.120	163	1.083	1.182	29.2	200
29	54.820	161	1.047	1.185	23.6	130
29	55.620	148	1.032	1.185	23.6	100
26	52.130	147	1.062	1.182	29.2	160
1408	38.550	3840	1.801	0.838	33.9	216
1260	39.230	3798	1.711	0.851	36.6	218
1295	39.480	3647	1.722	0.831	34	218
1184	36.780	3220	1.779	0.798	36.4	238
886	39.780	3212	1.536	0.806	40.3	219
1246	36.190	3200	1.852	0.91	39.6	250
1102	37.670	3187	1.707	0.861	40.3	228
1018	40.220	3184	1.647	0.865	31.2	226
1186	37.380	3172	1.753	0.825	37.6	230
1439	36.240	2946	1.946	0.924	36.9	240
1008	38.650	2944	1.65	0.841	37.5	230
1016	38.840	2768	1.686	0.942	36.8	218
941	39.220	2568	1.677	1.036	36.6	230
963	40.580	2509	1.572	0.865	36.8	220
948	40.780	2482	1.619	1.061	37.2	229
816	40.70	2425	1.571	0.873	31.3	250
889	39.0	2417	1.602	0.899	39.6	220
882	40.570	2310	1.62	1.063	35.2	229
765	40.920	2254	1.556	0.923	31.8	243
737	40.820	2061	1.533	0.936	34.5	234
523	41.350	1920	1.422	0.838	35.6	250
554	43.390	1719	1.416	0.975	31.7	216
631	41.970	1625	1.489	1.047	33.5	244
583	42.230	1591	1.475	1.054	32.2	239
537	43.610	1490	1.424	0.989	29.4	239
554	40.580	1430	1.478	0.958	35.8	226
490	45.080	1401	1.342	0.959	31.7	212
390	42.170	1345	1.364	0.923	36.3	254
439	45.210	1325	1.345	1.145	32.1	213

364	46.590	1261	1.29	0.987	28.4	215
405	45.940	1207	1.322	1.079	29.7	212
457	41.960	1197	1.412	1.05	36	220
406	44.210	1179	1.334	1.048	34.5	220
446	44.080	1141	1.335	0.98	35.4	190
409	45.850	1110	1.328	1.087	29.5	234
408	44.980	1104	1.346	1.069	30.2	232
392	45.170	1065	1.305	1.061	34.2	213
393	44.650	1062	1.34	1.09	32	234
333	45.430	1030	1.322	1.055	28.2	230
343	46.030	994	1.301	1.16	30.6	230
242	47.040	901	1.24	1.12	30.1	235
265	47.120	710	1.252	1.144	29.4	216
209	48.260	601	1.216	1.29	29	218
141	51.190	545	1.125	1.072	27.5	155
266	48.680	518	1.163	1.192	37.3	105
127	53.060	515	1.096	1.025	25.1	120
141	51.880	508	1.11	1.072	27.5	130
158	50.030	477	1.169	1.308	27.1	220
168	49.130	444	1.173	1.367	30.5	205
62	52.510	421	1.045	0.875	31.6	170
104	52.060	408	1.098	1.126	27.4	160
168	50.20	392	1.148	1.367	30.5	165
79	52.950	370	1.099	1.146	23.5	185
100	51.390	368	1.124	1.247	26	205
168	51.220	343	1.125	1.367	30.5	125
74	52.570	331	1.078	1.093	27.4	160
79	53.890	327	1.08	1.146	23.5	145
74	53.510	293	1.059	1.093	27.4	120
103	52.480	290	1.108	1.335	25.4	155

Appendix II: GMDH Matlab Script Code

Main Function :

```
clc
clf
clear all;%Clears all variables and other classes of data too.
close all;
tic
%to reduce the risk of confusing errors.
%
% Step (1) Reading the input file
% ================================
% Loads data and prepares it for a neural network.
%ndata= xlsread('all_data.xls');
ndata= xlsread('main_data.xlsx');
%50% of data will be used for training
%25% of data will be used for cross-validation
%25% of data will be used for testing
for i=1:189
    atr(i,:)=ndata(i,:);
end
for i=190:283
    aval(i-189,:)=ndata(i,:);
end
%
for i=284:length(ndata)
    atest(i-283,:)=ndata(i,:);
end
Ytr=atr(:,1);
Xtr=atr(:,2:5);
Xtst=atest(:,2:5);
Ytst=atest(:,1);
Yv=aval(:,1);
Xv=aval(:,2:5);
[model, time] = gmdhbuild(Xtr, Ytr, 3, 1, 2, 0, 2, 1, 0.9, Xv, Yv,1);
gmdheq(model, 3);
[Yqtst] = gmdhpredict(model, Xtst);
[Yqval] = gmdhpredict(model, Xv);
[Yqtr] = gmdhpredict(model, Xtr);
[MSE, RMSE, RRMSE, R2] = gmdhtest(model, Xtst, Ytst);

% Evaluating Relative Error for training set:
%============================================
Et1=(Ytr-Yqtr)./Ytr*100;
[q,z] = size(Et1);
figure
plot(Ytst,Yqtst,'sg')
grid off
set(gcf, 'color', 'white')
axis square

title('Predicted Gas Oil Ratio vs. Measured Gas Oil Ratio');
xlabel('Measured Gas Oil Ratio GOR "ft3/STB"');
ylabel('Predicted Gas Oil Ratio GOR "ft3/STB"')
legend('Training set', 'location', 'Northwest')
% Addding Reference Line with 45 degree slope
line([0 ; 2000],[0 ; 2000])
%HINT: Select the y-value based on your data limits
hold
```

```
% Evaluating the correlation coefficient for training set:
% ========================================================
Rt1=corrcoef(Yqtr,Ytr);
Rt11=min(Rt1(:,1));
gtext(['correlation coefficient = (' num2str(Rt11) ')']);
hold

line([0 ; 2000],[0 ; 2000])


Ev1=(Yqval-Yv)./Yqval*100;
[m,n] = size(Ev1);
figure

plot(Yv,Yqval,'ob')
grid off
set(gcf, 'color', 'white')
axis square
title('Predicted Gas Oil Ratio vs. Measured Gas Oil Ratio');
xlabel('Measured Gas Oil Ratio GOR "ft3/STB"');
ylabel('Predicted Gas Oil Ratio GOR "ft3/STB"')
legend('Validation set', 'location', 'Northwest')
% Addding Reference Line with 45 degree slope
line([0 ; 2000],[0 ; 2000])
%HINT: Select the y-value based on your data limits

Rv1=corrcoef(Yqval,Yv);
Rv11=min(Rv1(:,1));
gtext(['correlation coefficient = (' num2str(Rv11) ')']);
hold

% Evaluating Relative Error for testing set:
%===========================================
Ett1=(Ytst-Yqtst)./Ytst*100;
[m,n] = size(Ett1);
figure
%
plot(Ytst,Yqtst,'vb')
grid off
set(gcf, 'color', 'white')
axis square

title('Predicted Gas Oil Ratio vs.Measured Gas Oil Ratio');
xlabel('Measured Gas Oil Ratio GOR "ft3/STB"');
ylabel('Predicted Gas Oil Ratio GOR "ft3/STB"')
legend('Testing set', 'location', 'Northwest')
% Addding Reference Line with 45 degree slope
line([0 ; 2000],[0 ; 2000])
%HINT: Select the y-value based on your data limits

% Evaluating the correlation coefficient for testing set:
% ========================================================
Rtt1=corrcoef(Yqtst,Ytst);
Rtt11=min(Rtt1(:,1));
gtext(['correlation coefficient = (' num2str(Rtt11) ')']);
hold
% plotting the histogram of the errors for training set:
% ========================================================
figure
histfit(Et1,10)
```

```
%hist(Et1,10)
h = findobj(gca,'Type','patch');
set(h,'FaceColor','w','EdgeColor','k')
title('Error Distribution for Training Set (Polynomial GMDH Model)');
legend('Training set')
xlabel('Error');
ylabel('Frequency')
set(gcf, 'color', 'white')
hold

% plotting the histogram of the errors for validation set:
% ========================================================
figure
histfit(Ev1,10)
%hist(Ev1,10)
h = findobj(gca, 'Type', 'patch');
set(h,'FaceColor','w','EdgeColor','k')
title('Error Distribution for Validation Set (Polynomial GMDH
Model)');
legend('Validation set')
xlabel('Error');
ylabel('Frequency')
set(gcf, 'color', 'white')
hold

% plotting the histogram of the errors for testing set:
% =====================================================
figure
histfit(Ett1,10)
%hist(Ett1,10)
h = findobj(gca,'Type','patch');
set(h,'FaceColor','w','EdgeColor','k')
title('Error Distribution for Testing Set (Polynomial GMDH Model)');
legend('Testing set')
xlabel('Error');
ylabel('Frequency')
set(gcf, 'color', 'white')
hold
% Estimating the residuals for training set:
% ==========================================
figure
Errort1 = Yqtr-Ytr;
plot(Errort1,':ro');
grid off
set(gcf, 'color', 'white')
title('Error Distribution for Training Set (Polynomial GMDH Model)')
legend('Training Set')
xlabel('Data Point No')
ylabel('Errors')
hold
% Estimating the residuals for validation set:
% ============================================
figure
Errorv1 = Yqval-Yv;
plot(Errorv1,':ro');
grid off
set(gcf, 'color', 'white')
title('Residual Graph for Validation Set (Polynomial GMDH Model)')
legend('Validation Set')
xlabel('Data Point No')
ylabel('Errors')
```

```
hold
% Estimating the residuals for testing set:
% ========================================
figure
Errortt1 = Yqtst-Ytst;
plot(Errortt1,':ro');
grid off
set(gcf, 'color', 'white')
title('Residual Graph for Testing Set (Polynomial GMDH Model)')
legend('Testing Set')
xlabel('Data Point No')
ylabel('Errors')

% ********************
% STATISTICAL ANALYSIS:
% ********************
% Training set:
% =============
% Determining the Maximum Absolute Percent Relative Error
MaxErrt1 = max(abs(Et1));

% Evaluating the average error
Etavg1 = 1/z*sum(Et1);

% Evaluating the standard deviation
STDT1 = std(Errort1);

% Determining the Minimum Absolute Percent Relative Error
MinErrt1 = min(abs(Et1));

% Evaluating Average Absolute Percent Relative Error
% ================================================
AAPET1 = sum(abs(Et1))/q;

% Evaluating Average Percent Relative Error
% =========================================
APET1 = 1/q*sum(Et1);

% Evaluating Root Mean Square
% ===========================
RMSET1 = sqrt(sum(abs(Et1).^2)/q);

% Validation set:
% ===============
% Determining the Maximum Absolute Percent Relative Error
MaxErrv1 = max(abs(Ev1));

% Determining the Minimum Absolute Percent Relative Error
MinErrv1 = min(abs(Ev1));

% Evaluating the average error
Evavg1 = 1/m*sum(Ev1);

% Evaluating the standard deviation
STDV1 = std(Errorv1);

%
% Evaluating Average Absolute Percent Relative Error
% ================================================
```

```
AAPEV1 = sum(abs(Ev1))/m;

% Evaluating Average Percent Relative Error
% ========================================
APEV1 = 1/m*sum(Ev1);

% Evaluating Root Mean Square
% ===========================
RMSEV1 = sqrt(sum(abs(Ev1).^2)/m);

% Testing set:
% ============
% Determining the Maximum Absolute Percent Relative Error
MaxErrtt1 = max(abs(Ett1));

% Determining the Minimum Absolute Percent Relative Error
MinErrtt1 = min(abs(Ett1));

% Evaluating the average error
Ettavg1 = 1/m*sum(Ett1);

% Evaluating the standard deviation
STDTT1 = std(Errortt1);

% Evaluating Average Absolute Percent Relative Error
% ================================================
AAPETT1 = sum(abs(Ett1))/m;

% Evaluating Average Percent Relative Error
% =========================================
APETT1 = 1/m*sum(Ett1);

% Evaluating Root Mean Square
% ===========================
RMSETT1 = sqrt(sum(abs(Ett1).^2)/m);
```

Build Function

```
function [model, time] = gmdhbuild(Xtr, Ytr, maxNumInputs,
inputsMore, ...
maxNumNeurons, decNumNeurons, p, critNum, delta, Xv, Yv,
verbose)
% GMDHBUILD
% Builds a GMDH-type polynomial neural network using a
simple
% layer-by-layer approach
%
% Call
%   [model, time] = gmdhbuild(Xtr, Ytr, maxNumInputs,
inputsMore, maxNumNeurons,
%                   decNumNeurons, p, critNum, delta, Xv,
Yv, verbose)
%   [model, time] = gmdhbuild(Xtr, Ytr, maxNumInputs,
inputsMore, maxNumNeurons,
```

```
%                   decNumNeurons, p, critNum, delta, Xv,
Yv)
%   [model, time] = gmdhbuild(Xtr, Ytr, maxNumInputs,
inputsMore, maxNumNeurons,
%                   decNumNeurons, p, critNum, delta)
%   [model, time] = gmdhbuild(Xtr, Ytr, maxNumInputs,
inputsMore, maxNumNeurons,
%                   decNumNeurons, p, critNum)
%   [model, time] = gmdhbuild(Xtr, Ytr, maxNumInputs,
inputsMore, maxNumNeurons,
%                   decNumNeurons, p)
%   [model, time] = gmdhbuild(Xtr, Ytr, maxNumInputs,
inputsMore, maxNumNeurons,
%                   decNumNeurons)
%   [model, time] = gmdhbuild(Xtr, Ytr, maxNumInputs,
inputsMore, maxNumNeurons)
%   [model, time] = gmdhbuild(Xtr, Ytr, maxNumInputs,
inputsMore)
%   [model, time] = gmdhbuild(Xtr, Ytr, maxNumInputs)
%   [model, time] = gmdhbuild(Xtr, Ytr)
%
% Input
% Xtr, Ytr     : Training data points (Xtr(i,:), Ytr(i)),
i = 1,...,n
% maxNumInputs : Maximum number of inputs for individual
neurons - if set
%                to 3, both 2 and 3 inputs will be tried
(default = 2)
% inputsMore   : Set to 0 for the neurons to take inputs
only from the
%                preceding layer, set to 1 to take inputs
also from the
%                original input variables (default = 1)
% maxNumNeurons: Maximal number of neurons in a layer
(default = equal to
%                the number of the original input
variables)
% decNumNeurons: In each following layer decrease the
number of allowed
%                neurons by decNumNeurons until the
number is equal to 1
%                (default = 0)
% p            : Degree of polynomials in neurons
(allowed values are 2 and
%                3) (default = 2)
% critNum      : Criterion for evaluation of neurons and
for stopping.
%                In each layer only the best neurons
(according to the
%                criterion) are retained, and the rest
are discarded.
```

```
%                 (default = 2)
%                 0 = use validation data (Xv, Yv)
%                 1 = use validation data (Xv, Yv) as well
as training data
%                 2 = use Corrected Akaike's Information
Criterion (AICC)
%                 3 = use Minimum Description Length (MDL)
%                 Note that both choices 0 and 1
correspond to the so called
%                 "regularity criterion".
% delta         : How much lower the criterion value of
the network's new
%                 layer must be comparing the the
network's preceding layer
%                 (default = 0, which means that new
layers will be added as
%                 long as the value gets better (smaller))
% Xv, Yv        : Validation data points (Xv(i,:), Yv(i)),
i = 1,...,nv
%                 (used when critNum is equal to either 0
or 1)
% verbose       : Set to 0 for no verbose (default = 1)
%
% Output
% model         : GMDH model - a struct with the following
elements:
%    numLayers     : Number of layers in the network
%    d             : Number of input variables in the
training data set
%    maxNumInputs  : Maximal number of inputs for neurons
%    inputsMore    : See argument "inputsMore"
%    maxNumNeurons : Maximal number of neurons in a layer
%    p             : See argument "p"
%    critNum       : See argument "critNum"
%    layer         : Full information about each layer
(number of neurons,
%                    indexes of inputs for neurons,
matrix of exponents for
%                    polynomial, polynomial coefficients)
%                    Note that the indexes of inputs are
in range [1..d] if
%                    an input is one of the original
input variables, and
%                    in range [d+1..d+maxNumNeurons] if
an input is taken
%                    from a neuron in the preceding layer.
% time          : Execution time (in seconds)
%
% Please give a reference to the software web page in any
publication
```

```
% describing research performed using the software e.g.,
like this:
% Jekabsons G. GMDH-type Polynomial Neural Networks for
Matlab, 2010,
% available at http://www.cs.rtu.lv/jekabsons/

% This source code is tested with Matlab version 7.1
(R14SP3).

%
================================================================
% GMDH-type polynomial neural network
% Version: 1.5
% Date: June 2, 2011
% Author: Gints Jekabsons (gints.jekabsons@rtu.lv)
% URL: http://www.cs.rtu.lv/jekabsons/
%
% Copyright (C) 2009-2011  Gints Jekabsons
%
% This program is free software: you can redistribute it
and/or modify
% it under the terms of the GNU General Public License as
published by
% the Free Software Foundation, either version 2 of the
License, or
% (at your option) any later version.
%
% This program is distributed in the hope that it will be
useful,
% but WITHOUT ANY WARRANTY; without even the implied
warranty of
% MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.
See the
% GNU General Public License for more details.
%
% You should have received a copy of the GNU General
Public License
% along with this program. If not, see
<http://www.gnu.org/licenses/>.
%
================================================================

if nargin < 2
    error('Too few input arguments.');
end

[n, d] = size(Xtr);
[ny, dy] = size(Ytr);
if (n < 2) || (d < 2) || (ny ~= n) || (dy ~= 1)
```

```
    error('Wrong training data sizes.');
end


if (d < maxNumInputs)
    error('Numbet of input variables in the data is lower
than the number of inputs for individual neurons.');
end
if nargin < 4
    inputsMore = 1;
end
if (nargin < 5) || (maxNumNeurons <= 0)
    maxNumNeurons = d;
end
if maxNumNeurons > d * 2
    error('Too many neurons in a layer. Maximum is two
times the number of input variables.');
end
if maxNumNeurons < 1
    error('Too few neurons in a layer. Minimum is 1.');
end
if (nargin < 6) || (decNumNeurons < 0)
    decNumNeurons = 0;
end
if nargin < 7
    p = 2;
elseif (p ~= 2) && (p ~= 3)
    error('Degree of individual neurons should be 2 or
3.');
end
if nargin < 8
    critNum = 2;
end
if any(critNum == [0,1,2,3]) == 0
    error('Only four values for critNum are available
(0,1 - use validation data; 2 - AICC; 3 - MDL).');
end
if nargin < 9
    delta = 0;
end
if (nargin < 11) && (critNum <= 1)
    error('Evaluating the models in validation data
requires validation data set.');
end
if (nargin >= 11) && (critNum <= 1)
    [nv, dv] = size(Xv);
    [nvy, dvy] = size(Yv);
    if (nv < 1) || (dv ~= d) || (nvy ~= nv) || (dvy ~= 1)
        error('Wrong validation data sizes.');
    end
end
```

```
if nargin < 12
    verbose = 1;
end

ws = warning('off');
if verbose ~= 0
    fprintf('Building GMDH-type neural network...\n');
end
tic;

if p == 2
    numTermsReal = 6 + 4 * (maxNumInputs == 3); %6 or 10
terms
else
    numTermsReal = 10 + 10 * (maxNumInputs == 3); %10 or
20 terms
end

Xtr(:, d+1:d+maxNumNeurons) = zeros(n, maxNumNeurons);
if critNum <= 1
     Xv(:, d+1:d+maxNumNeurons) = zeros(nv,
maxNumNeurons);
end

%start the main loop and create layers
model.numLayers = 0;
while 1

    if verbose ~= 0
        fprintf('Building layer #%d...\n',
model.numLayers + 1);
    end

    layer(model.numLayers + 1).numNeurons = 0;
    modelsTried = 0;
    layer(model.numLayers + 1).coefs =
zeros(maxNumNeurons, numTermsReal);

    for numInputsTry = maxNumInputs:-1:2

        %create matrix of exponents for polynomials
        if p == 2
            numTerms = 6 + 4 * (numInputsTry == 3); %6 or
10 terms
            if numInputsTry == 2
                r = [0,0;0,1;1,0;1,1;0,2;2,0];
            else
                r =
[0,0,0;0,0,1;0,1,0;1,0,0;0,1,1;1,0,1;1,1,0;0,0,2;0,2,0;2,
0,0];
            end
```

```
        else
            numTerms = 10 + 10 * (numInputsTry == 3); %10
or 20 terms
            if numInputsTry == 2
                r =
[0,0;0,1;1,0;1,1;0,2;2,0;1,2;2,1;0,3;3,0];
            else
                r =
[0,0,0;0,0,1;0,1,0;1,0,0;0,1,1;1,0,1;1,1,0;0,0,2;0,2,0;2,
0,0; ...

1,1,1;0,1,2;0,2,1;1,0,2;1,2,0;2,0,1;2,1,0;0,0,3;0,3,0;3,0
,0];
            end
        end

        %create matrix of all combinations of inputs for
neurons
        if model.numLayers == 0
            combs = nchoosek(1:1:d, numInputsTry);
        else
            if inputsMore == 1
                combs = nchoosek([1:1:d
d+1:1:d+layer(model.numLayers).numNeurons], numInputsTry);
            else
                combs =
nchoosek(d+1:1:d+layer(model.numLayers).numNeurons,
numInputsTry);
            end
        end
        %delete all combinations in which none of the
inputs are from the preceding layer
        if model.numLayers > 0
            i = 1;
            while i <= size(combs,1)
                if all(combs(i,:) <= d)
                    combs(i,:) = [];
                else
                    i = i + 1;
                end
            end
        end

        makeEmpty = 1;

        %try all the combinations of inputs for neurons
        for i = 1 : size(combs,1)

            %create matrix for all polynomial terms
            Vals = ones(n, numTerms);
            if critNum <= 1
```

```
                Valsv = ones(nv, numTerms);
            end
            for idx = 2 : numTerms
                bf = r(idx, :);
                t = bf > 0;
                tmp = Xtr(:, combs(i,t)) .^ bf(ones(n, 1),
t);
                if critNum <= 1
                    tmpv = Xv(:, combs(i,t)) .^
bf(ones(nv, 1), t);
                end
                if size(tmp, 2) == 1
                    Vals(:, idx) = tmp;
                    if critNum <= 1
                        Valsv(:, idx) = tmpv;
                    end
                else
                    Vals(:, idx) = prod(tmp, 2);
                    if critNum <= 1
                        Valsv(:, idx) = prod(tmpv, 2);
                    end
                end
            end

            %calculate coefficients and evaluate the
network
            coefs = (Vals' * Vals) \ (Vals' * Ytr);
            modelsTried = modelsTried + 1;
            if ~isnan(coefs(1))
                predY = Vals * coefs;
                if critNum <= 1
                    predYv = Valsv * coefs;
                    if critNum == 0
                        crit = sqrt(mean((predYv -
Yv).^2));
                    else
                        crit = sqrt(mean([(predYv -
Yv).^2; (predY - Ytr).^2]));
                    end
                else
                    comp = complexity(layer,
model.numLayers, maxNumNeurons, d, combs(i,:)) +
size(coefs, 2);
                    if critNum == 2 %AICC
                        if (n-comp-1 > 0)
                            crit = n*log(mean((predY -
Ytr).^2)) + 2*comp + 2*comp*(comp+1)/(n-comp-1);
                        else
                            coefs = NaN;
                        end
                    else %MDL
```

```
                        crit = n*log(mean((predY -
Ytr).^2)) + comp*log(n);
                    end
                end
            end

            if ~isnan(coefs(1))
                %add the neuron to the layer if
                %1) the layer is not full;
                %2) the new neuron is better than an
existing worst one.
                maxN = maxNumNeurons - model.numLayers *
decNumNeurons;
                if maxN < 1, maxN = 1; end;
                if layer(model.numLayers + 1).numNeurons
< maxN
                    %when the layer is not yet full
                    if (maxNumInputs == 3) &&
(numInputsTry == 2)
                        layer(model.numLayers +
1).coefs(layer(model.numLayers + 1).numNeurons+1, :) =
[coefs' zeros(1,4+6*(p == 3))];
                        layer(model.numLayers +
1).inputs(layer(model.numLayers + 1).numNeurons+1, :) =
[combs(i, :) 0];
                    else
                        layer(model.numLayers +
1).coefs(layer(model.numLayers + 1).numNeurons+1, :) =
coefs;
                        layer(model.numLayers +
1).inputs(layer(model.numLayers + 1).numNeurons+1, :) =
combs(i, :);
                    end
                    layer(model.numLayers +
1).comp(layer(model.numLayers + 1).numNeurons+1) =
length(coefs);
                    layer(model.numLayers +
1).crit(layer(model.numLayers + 1).numNeurons+1) = crit;
                    layer(model.numLayers +
1).terms(layer(model.numLayers + 1).numNeurons+1).r = r;
                    if makeEmpty == 1
                        Xtr2 = [];
                        if critNum <= 1
                            Xv2 = [];
                        end
                        makeEmpty = 0;
                    end
                    Xtr2(:, layer(model.numLayers +
1).numNeurons+1) = predY;
                    if critNum <= 1
```

```
                        Xv2(:, layer(model.numLayers +
1).numNeurons+1) = predYv;
                    end
                    if (layer(model.numLayers +
1).numNeurons == 0) || ...
                       (layer(model.numLayers +
1).crit(worstOne) < crit)
                        worstOne = layer(model.numLayers
+ 1).numNeurons + 1;
                    end
                    layer(model.numLayers + 1).numNeurons
= layer(model.numLayers + 1).numNeurons + 1;
                else
                    %when the layer is already full
                    if (layer(model.numLayers +
1).crit(worstOne) > crit)
                        if (maxNumInputs == 3) &&
(numInputsTry == 2)
                            layer(model.numLayers +
1).coefs(worstOne, :) = [coefs' zeros(1,4+6*(p == 3))];
                            layer(model.numLayers +
1).inputs(worstOne, :) = [combs(i, :) 0];
                        else
                            layer(model.numLayers +
1).coefs(worstOne, :) = coefs;
                            layer(model.numLayers +
1).inputs(worstOne, :) = combs(i, :);
                        end
                        layer(model.numLayers +
1).comp(worstOne) = length(coefs);
                        layer(model.numLayers +
1).crit(worstOne) = crit;
                        layer(model.numLayers +
1).terms(worstOne).r = r;
                        Xtr2(:, worstOne) = predY;
                        if critNum <= 1
                            Xv2(:, worstOne) = predYv;
                        end
                        [dummy, worstOne] =
max(layer(model.numLayers + 1).crit);
                    end
                end
            end

        end

    end

    if verbose ~= 0
        fprintf('Neurons tried in this layer: %d\n',
modelsTried);
```

```
        fprintf('Neurons included in this layer: %d\n',
layer(model.numLayers + 1).numNeurons);
        if critNum <= 1
            fprintf('RMSE in the validation data of the
best neuron: %f\n', min(layer(model.numLayers + 1).crit));
        else
            fprintf('Criterion value of the best
neuron: %f\n', min(layer(model.numLayers + 1).crit));
        end
    end

    %stop the process if there are too few neurons in the
new layer
    if ((inputsMore == 0) && (layer(model.numLayers +
1).numNeurons < 2)) || ...
       ((inputsMore == 1) && (layer(model.numLayers +
1).numNeurons < 1))
        if (layer(model.numLayers + 1).numNeurons > 0)
            model.numLayers = model.numLayers + 1;
        end
        break
    end

    %if the network got "better", continue the process
    if (layer(model.numLayers + 1).numNeurons > 0) && ...
       ((model.numLayers == 0) || ...
        (min(layer(model.numLayers).crit) -
min(layer(model.numLayers + 1).crit) >
delta) ) %(min(layer(model.numLayers + 1).crit) <
min(layer(model.numLayers).crit)) )
        model.numLayers = model.numLayers + 1;
    else
        if model.numLayers == 0
            warning(ws);
            error('Failed.');
        end
        break
    end

    %copy the output values of this layer's neurons to
the training
    %data matrix
    Xtr(:, d+1:d+layer(model.numLayers).numNeurons) =
Xtr2;
    if critNum <= 1
        Xv(:, d+1:d+layer(model.numLayers).numNeurons) =
Xv2;
    end

end
```

```
model.d = d;
model.maxNumInputs = maxNumInputs;
model.inputsMore = inputsMore;
model.maxNumNeurons = maxNumNeurons;
model.p = p;
model.critNum = critNum;

%only the neurons which are actually used (directly or
indirectly) to
%compute the output value may stay in the network
[dummy best] = min(layer(model.numLayers).crit);
model.layer(model.numLayers).coefs(1,:) =
layer(model.numLayers).coefs(best,:);
model.layer(model.numLayers).inputs(1,:) =
layer(model.numLayers).inputs(best,:);
model.layer(model.numLayers).terms(1).r =
layer(model.numLayers).terms(best).r;
model.layer(model.numLayers).numNeurons = 1;
if model.numLayers > 1
    for i = model.numLayers-1:-1:1 %loop through all the
layers
        model.layer(i).numNeurons = 0;
        for k = 1 : layer(i).numNeurons %loop through all
the neurons in this layer
            newNum = 0;
            for j = 1 : model.layer(i+1).numNeurons %loop
through all the neurons which will stay in the next layer
                for jj = 1 : maxNumInputs %loop through
all the inputs
                    if k == model.layer(i+1).inputs(j,jj)
- d
                        if newNum == 0
                            model.layer(i).numNeurons =
model.layer(i).numNeurons + 1;

model.layer(i).coefs(model.layer(i).numNeurons,:) =
layer(i).coefs(k,:);

model.layer(i).inputs(model.layer(i).numNeurons,:) =
layer(i).inputs(k,:);

model.layer(i).terms(model.layer(i).numNeurons).r =
layer(i).terms(k).r;
                            newNum =
model.layer(i).numNeurons + d;
                            model.layer(i+1).inputs(j,jj)
= newNum;
                        else
                            model.layer(i+1).inputs(j,jj)
= newNum;
                        end
```

```
                        break
                    end
                end
            end
        end
    end
end

time = toc;
warning(ws);

if verbose ~= 0
    fprintf('Done.\n');
    used = zeros(d,1);
    for i = 1 : model.numLayers
        for j = 1 : d
            if any(any(model.layer(i).inputs == j))
                used(j) = 1;
            end
        end
    end
    fprintf('Number of layers: %d\n', model.numLayers);
    fprintf('Number of used input variables: %d\n',
sum(used));
    fprintf('Execution time: %0.2f seconds\n', time);
end

return

%====================  Auxiliary functions
====================

function [comp] = complexity(layer, numLayers,
maxNumNeurons, d, connections)
%calculates the complexity of the network given output
neuron's connections
%(it is assumed that the complexity of a network is equal
to the number of
%all polynomial terms in all it's neurons which are
actually connected
%(directly or indirectly) to network's output)
comp = 0;
if numLayers == 0
    return
end
c = zeros(numLayers, maxNumNeurons);
for i = 1 : numLayers
    c(i, :) = layer(i).comp(:)';
end
%{
%unvectorized version:
```

```
for j = 1 : length(connections)
    if connections(j) > d
        comp = comp + c(numLayers, connections(j) - d);
        c(numLayers, connections(j) - d) = -1;
    end
end
%}
ind = connections > d;
if any(ind)
    comp = comp + sum(c(numLayers, connections(ind) - d));
    c(numLayers, connections(ind) - d) = -1;
end
%{
%unvectorized version:
for i = numLayers-1:-1:1
    for j = 1 : layer(i).numNeurons
        for k = 1 : layer(i+1).numNeurons
            if (c(i+1, k) == -1) && (c(i, j) > -1) && ...
               any(layer(i+1).inputs(k,:) == j + d)
                comp = comp + c(i, j);
                c(i, j) = -1;
            end
        end
    end
end
%}
for i = numLayers-1:-1:1
        for k = 1 : layer(i+1).numNeurons
            if c(i+1, k) == -1
                inp = layer(i+1).inputs(k,:);
                used = inp > d;
                if any(used)
                    ind = inp(used) - d;
                    ind = ind(c(i, ind) > -1);
                    if ~isempty(ind)
                        comp = comp + sum(c(i, ind));
                        c(i, ind) = -1;
                    end
                end
            end
        end
end
return
```

Equation Function:

```
function gmdheq(model, precision)
% gmdheq
% Outputs the equations of GMDH model.
%
% Call
%   gmdheq(model, precision)
%   gmdheq(model)
%
% Input
%   model       : GMDH-type model
%   precision    : Number of digits in the model coefficients
%                (default = 15)

% This source code is tested with Matlab version 7.1 (R14SP3).

%
================================================================================
% GMDH-type polynomial neural network
% Version: 1.5
% Date: June 2, 2011
% Author: Gints Jekabsons (gints.jekabsons@rtu.lv)
% URL: http://www.cs.rtu.lv/jekabsons/
%
% Copyright (C) 2009-2011  Gints Jekabsons
```

```
%
% This program is free software: you can redistribute it and/or modify
% it under the terms of the GNU General Public License as published by
% the Free Software Foundation, either version 2 of the License, or
% (at your option) any later version.
%
% This program is distributed in the hope that it will be useful,
% but WITHOUT ANY WARRANTY; without even the implied warranty of
% MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
% GNU General Public License for more details.
%
% You should have received a copy of the GNU General Public License
% along with this program. If not, see <http://www.gnu.org/licenses/>.
%
================================================================================


if nargin < 1
   error('Too few input arguments.');
end
if (nargin < 2) || (isempty(precision))
   precision = 15;
end

if model.numLayers > 0
   p = ['%.' num2str(precision) 'g'];
   fprintf('Number of layers: %d\n', model.numLayers);
```

```
    for i = 1 : model.numLayers %loop through all the layers
        fprintf('Layer #%d\n', i);
        fprintf('Number of neurons: %d\n', model.layer(i).numNeurons);
        for j = 1 : model.layer(i).numNeurons %loop through all the neurons in the ith
layer
            [terms inputs] = size(model.layer(i).terms(j).r); %number of terms and inputs
            if (i == model.numLayers)
                str = ['y = ' num2str(model.layer(i).coefs(j,1),p)];
            else
                str = ['x' num2str(j + i*model.d) ' = ' num2str(model.layer(i).coefs(j,1),p)];
            end
            for k = 2 : terms %loop through all the terms
                if model.layer(i).coefs(j,k) >= 0
                    str = [str ' +'];
                else
                    str = [str ' '];
                end
                str = [str num2str(model.layer(i).coefs(j,k),p)];
                for kk = 1 : inputs %loop through all the inputs
                    if (model.layer(i).terms(j).r(k,kk) > 0)
                        for kkk = 1 : model.layer(i).terms(j).r(k,kk)
                            if (model.layer(i).inputs(j,kk) <= model.d)
                                str = [str '*x' num2str(model.layer(i).inputs(j,kk))];
                            else
                                str = [str '*x' num2str(model.layer(i).inputs(j,kk) + (i-2)*model.d)];
                            end
```

```
                end
            end
        end
    end
    disp(str);
  end
 end
else
 disp('The network has zero layers.');
end

return
```

Prediction Function

```
function Yq = gmdhpredict(model, Xq)
% GMDHPREDICT
% Predicts output values for the given query points Xq using a GMDH model
%
% Call
%   [Yq] = gmdhpredict(model, Xq)
%
% Input
% model    : GMDH model
% Xq       : Inputs of query data points (Xq(i,:)), i = 1,...,nq
```

```
%
% Output
% Yq        : Predicted outputs of query data points (Yq(i)), i = 1,...,nq


% This source code is tested with Matlab version 7.1 (R14SP3).


%
================================================================================
% GMDH-type polynomial neural network
% Version: 1.5
% Date: June 2, 2011
% Author: Gints Jekabsons (gints.jekabsons@rtu.lv)
% URL: http://www.cs.rtu.lv/jekabsons/
%
% Copyright (C) 2009-2011  Gints Jekabsons
%
% This program is free software: you can redistribute it and/or modify
% it under the terms of the GNU General Public License as published by
% the Free Software Foundation, either version 2 of the License, or
% (at your option) any later version.
%
% This program is distributed in the hope that it will be useful,
% but WITHOUT ANY WARRANTY; without even the implied warranty of
% MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
% GNU General Public License for more details.
%
```

```
% You should have received a copy of the GNU General Public License
% along with this program. If not, see <http://www.gnu.org/licenses/>.
%
================================================================================


if nargin < 2
   error('Too few input arguments.');
end
if model.d ~= size(Xq, 2)
   error('The matrix should have the same number of columns as the matrix with which the network was built.');
end

[n, d] = size(Xq);
Yq = zeros(n, 1);

for q = 1 : n
   for i = 1 : model.numLayers
      if i ~= model.numLayers
         Xq_tmp = zeros(1, model.layer(i).numNeurons);
      end
      for j = 1 : model.layer(i).numNeurons

         %create matrix for all polynomial terms
         numTerms =  size(model.layer(i).terms(j).r,1);
         Vals = ones(numTerms,1);
```

```
            for idx = 2 : numTerms
               bf = model.layer(i).terms(j).r(idx, :);
               t = bf > 0;
               tmp = Xq(q, model.layer(i).inputs(j,t)) .^ bf(1, t);
               if size(tmp, 2) == 1
                  Vals(idx,1) = tmp;
               else
                  Vals(idx,1) = prod(tmp, 2);
               end
            end

            %predict output value
            predY = model.layer(i).coefs(j,1:numTerms) * Vals;
            if i ~= model.numLayers
               %Xq(q, d+j) = predY;
               Xq_tmp(j) = predY;
            else
               Yq(q) = predY;
            end
         end
         if i ~= model.numLayers
            Xq(q, d+1:d+model.layer(i).numNeurons) = Xq_tmp;
         end
      end
   end
end
return
```

Testing Function

```
function [MSE, RMSE, RRMSE, R2] = gmdhtest(model, Xtst, Ytst)
% GMDHTEST
% Tests a GMDH-type network model on a test data set (Xtst, Ytst)
%
% Call
%   [MSE, RMSE, RRMSE, R2] = gmdhtest(model, Xtst, Ytst)
%
% Input
% model    : GMDH model
% Xtst, Ytst: Test data points (Xtst(i,:), Ytst(i)), i = 1,...,ntst
%
% Output
% MSE      : Mean Squared Error
% RMSE     : Root Mean Squared Error
% RRMSE    : Relative Root Mean Squared Error
% R2       : Coefficient of Determination

% Copyright (C) 2009-2011  Gints Jekabsons

if nargin < 3
   error('Too few input arguments.');
end
if (size(Xtst, 1) ~= size(Ytst, 1))
   error('The number of rows in the matrix and the vector should be equal.');
```

```
end
if model.d ~= size(Xtst, 2)
   error('The matrix should have the same number of columns as the matrix with which the model was built.');
end
MSE = mean((gmdhpredict(model, Xtst) - Ytst) .^ 2);
RMSE = sqrt(MSE);
if size(Ytst, 1) > 1
   RRMSE = RMSE / std(Ytst, 1);
   R2 = 1 - MSE / var(Ytst, 1);
else
   RRMSE = Inf;
   R2 = Inf;
end
return
```