APPENDIX V: C++ CODE FOR TRAINING NEURAL NETWORK

```
1: #include <cstdlib>
2: #include <iostream>
3: #include <math.h>
4: #include "neural_variable.h"
5: #include <fstream>
6:
7: using namespace std ;
8:
9: ofstream let("init_weight.txt") ;
10: ofstream jacobian("jacobi.txt") ;
11: ofstream net("network.txt") ;
12:
13: void randomize_weights()
14: {
15: int i,j ;
16: int min=0 ;
17: int max=200 ;
18: int number ;
19:
20: srand((unsigned)time(0)) ;
21:
22: /////////////// range randomized weight ////////////
23: cout << " *************** RANDOM WEIGHTS ********************" << endl
<< endl;
24: let << " *************** RANDOM WEIGHTS ********************" << endl
<< endl;
25: ////////////////////////////////////////////////////////////////
26: ////////////////////////////////////////////////////////////////
27:
28: let<<"weight_1[" << input_var << "]= " ;
29: for(i=0 ; i<input_var ; i++) ////// column
30: {
31: number=( (( abs( rand() ) % (max-min+1) ) + min )) ;
32:
33: if(number > max )
34: number=max ;
35:
36: if(number < min )
37: number=min ;
38:
39: weight_1[i] = ((number / 100.0F) - 1) ;
40:
41: cout << weight_1[i] << " \t" ;
42: let << weight_1[i] << " \t\t" ;
43: }
44: cout<< endl ;
45: let<< endl ;
46:
47:
48:
49:
////////////////////////////////////////////////////////////////////////
/
50:
////////////////////////////////////////////////////////////////////////
/
51:
```

```
52: let<<"weight_2[" << input_var << "]= " ;
53: for(i=0 ; i<input_var ; i++) ////// column
54: {
55: number=( (( abs( rand() ) % (max-min+1) ) + min )) ;
56:
57: if(number > max )
58: number=max ;
59:
60: if(number < min )
61: number=min ;
62:
63: weight_2[i] = ((number / 100.0F) - 1) ;
64:
65: cout << weight_2[i] << " \t" ;
66: let << weight_2[i] << " \t\t" ;
67: }
68: cout<< endl ;
69: let<< endl ;
70:
71:
////////////////////////////////////////////////////////////////////////
72:
////////////////////////////////////////////////////////////////////////
73: let<<"weight_3["<< hidden_neuron <<"]= " ;
74: for(k=0 ; k<hidden_neuron ; k++) ////// column
75: {
76: number=( (( abs( rand() ) % (max-min+1) ) + min )) ;
77:
78: if(number > max )
79: number=max ;
80:
81: if(number < min )
82: number=min ;
83:
84: weight_3[k] = ((number / 100.0F) - 1) ;
85:
86: cout << weight_3[k] << " \t\t" ;
87: let << weight_3[k] << " \t\t" ;
88: }
89: cout<< endl ;
90: let<< endl ;
91:
92:
////////////////////////////////////////////////////////////////////////
93:
////////////////////////////////////////////////////////////////////////
94: let<<"weight_4["<< hidden_neuron <<"]= " ;
95:
96: for(k=0 ; k<hidden_neuron ; k++) ////// column
97: {
98: number=( (( abs( rand() ) % (max-min+1) ) + min )) ;
99:
100: if(number > max )
101: number=max ;
102:
103: if(number < min )
104: number=min ;
105:
106: weight_4[k] = ((number / 100.0F) - 1) ;
107:
108: cout << weight_4[k] << " \t" ;
```

```
109: let << weight_4[k] << " \t\t" ;
110: }
111: cout<< endl ;
112: let << endl ;
113:
114:
115:
////////////////////////////////////////////////////////////////////////////
116:
////////////////////////////////////////////////////////////////////////////
117:
118: let << " bias[" <<hidden_neuron + output_neuron << "]= " ;
119:
120: for(i=0 ; i<(hidden_neuron + output_neuron) ; i++) ////// column
121: {
122: number=( (( abs( rand() ) % (max-min+1) ) + min )) ;
123:
124: if(number > max )
125: number=max ;
126:
127: if(number < min )
128: number=min ;
129:
130: bias[i] = ((number / 100.0F) - 1) ;
131:
132: cout << bias[i] << " \t" ;
133: let << bias[i] << " \t\t" ;
134: }
135: cout<< endl ;
136: let<< endl ;
137: }
138:
139:
140:
141:
142: void nguyen_widrow()
143: {
144: randomize_weights() ;
145:
146: alpha= 1.0/input_var ;
147: gamma= pow(hidden_neuron,alpha ) ;
148: beta = ( 0.7 * gamma ) ;
149:
150: for(k=0 ; k<6 ; k++ )
151: {
152: /// eucl_sum_1 = eucl_sum_1 + pow(weight_1[k],2)
153: eucl_sum_1 += pow(weight_1[k],2) ;
154: /// eucl_sum_2 = eucl_sum_2 + pow(weight_2[k],2)
155: eucl_sum_2 += pow(weight_2[k],2) ;
156: }
157: eucl_sum_1 += ( pow(bias[0],2) + pow(weight_3[0],2) ) ;
158: eucl_sum_2 += ( pow(bias[1],2) + pow(weight_3[1],2) ) ;
159:
160: eucl_norm_1= sqrt( eucl_sum_1 ) ;
161: eucl_norm_2= sqrt( eucl_sum_2 ) ;
162:
163: let << endl;
164: let }
165: let<< "\tbeta= " << beta << endl ;
166: let<< "\thidden_norm_1= " << eucl_norm_1 << endl ;
167: let<< "\thidden_norm_2= " << eucl_norm_2 << endl << endl ;
```

```
168:
169: cout << " NGUYEN WIDROW WEIGHT " << endl << endl ;
170: let << "\t*** NGUYEN WIDROW INPUT ---> HIDDEN NEURON 1 WEIGHT*** "
<<endl<<endl;
171:
172:
173: for(n=0 ; n<input_var ; n++ )
174: {
175: i_weight_1[n] = ( beta * weight_1[n] )/( eucl_norm_1 ) ;
176:
177: cout << i_weight_1[n] << "\t " ;
178: let << i_weight_1[n] << "\t " ;
179: }
180: cout << endl ; let << endl ;
181:
182: let << "\t** NGUYEN WIDROW INPUT ---> HIDDEN NEURON 2 WEIGHT*** "<<
endl<<endl ;
183:
184:
185: for(n=0 ; n<(input_var) ; n++ )
186: {
187: i_weight_2[n] = ( beta * weight_2[n] )/( eucl_norm_2 ) ;
188:
189: cout << i_weight_2[n] << "\t " ;
190: let << i_weight_2[n] << "\t " ;
191: }
192: cout << endl ; let << endl ;
193:
194:
195: let << "\t******** NGUYEN WIDROW FEEDBACK WEIGHT****** " <<endl <<
endl ;
196:
197: for(n=0 ; n< (hidden_neuron) ; n++ )
198: {
199: i_weight_3[n] = ( beta * weight_3[n] )/( eucl_norm_2 ) ;
200:
201: cout << i_weight_3[n] << "\t " ;
202: let << i_weight_3[n] << "\t " ;
203: }
204: cout << endl ; let << endl ;
205:
206: let << "\t*** NGUYEN WIDROW HIDDEN ---> OUTPUT NEURON WEIGHT***
"<<endl<<endl;
207:
208: for(n=0 ; n<(hidden_neuron) ; n++)
209: {
210: cout << (i_weight_4[n]=weight_4[n]) << "\t" ;
211: let << (i_weight_4[n]=weight_4[n]) << "\t\t" ;
212: }
213:
214:
215: cout << endl ; let << endl ;
216:
217: cout << " NGUYEN WIDROW BIAS " << endl << endl ;
218: let << "************* NGUYEN WIDROW BIAS*************** " << endl <<
endl ;
219:
220: i_bias[0] = ( beta * bias[0] )/( eucl_norm_1 ) ;
221: i_bias[1] = ( beta * bias[1] )/( eucl_norm_2 ) ;
222:
223: for(n=0; n<(hidden_neuron+output_neuron) ; n++ )
```

```
{
if( n==2 )
{
i_bias[2] = bias[2] ;
}
cout << "\t " ;
let << "\t " ;

cout << i_bias[n] ;
let << i_bias[n] ;

}
cout << endl ; let << endl ;

}



void weight_summer()
{
nguyen_widrow() ;

////////////// for industrial input : output
////////////////////////////
if(n==1)
{
for(i=0 ; i<output_var ; i++ )
{
for( j=0 ; j<input_var ; j++ )
{
weighted_sum_1[i]= ( i_weight_1[j] * ind_input[i][j] ) +
( i_weight_3[0] * cal_output[i][0] ) + ( 1 * i_bias[0] ) ;
weighted_sum_2[i]= ( i_weight_2[j] * ind_input[i][j] ) +
( i_weight_3[1] * cal_output[i][0] ) + ( 1 * i_bias[1] ) ;
}
}
}


////////////// for residential input : output ///////////////

else if(n==2)
{
for(i=0 ; i<output_var ; i++ )
{
for( j=0 ; j<input_var ; j++ )
{
weighted_sum_1[i]= ( i_weight_1[j] * res_input[i][j] ) +
( i_weight_3[0] * cal_output[i][0] ) + ( 1 * i_bias[0] ) ;
weighted_sum_2[i]= ( i_weight_2[j] * res_input[i][j] ) +
( i_weight_3[1] * cal_output[i][0] ) + ( 1 * i_bias[1] ) ;
}
}
}

////////////// for commercial input : output ///////////////////////

else if(n==3)
{
for(i=0 ; i<output_var ; i++ )
```

```
{
for( j=0 ; j<input_var ; j++ )
{
weighted_sum_1[i]= ( i_weight_1[j] * com_input[i][j] ) +
( i_weight_3[0] * cal_output[i][0] ) + ( 1 * i_bias[0] ) ;
weighted_sum_2[i]= ( i_weight_2[j] * com_input[i][j] ) +
( i_weight_3[1] * cal_output[i][0] ) + ( 1 * i_bias[1] ) ;
}
}
}

}


void forward_pass()
{
weight_summer() ;

/////////////// WEIGHTED SUM PASSES THROUGH THE SIGMOID ACTIVATION
FUNCTION
////////

for(i=0 ; i<output_var ; i++)
{
net_out_1[i] = ( 1.0 / ( 1.0 + exp ( -weighted_sum_1[i]) ) ) ;
net_out_2[i] = ( 1.0 / ( 1.0 + exp ( -weighted_sum_2[i]) ) ) ;
}
}

void final_sum()
{
forward_pass() ;
let<< endl << endl ;
for(i=0 ; i<output_var ; i++)
{
out_sum_1[i] = ( i_weight_4[0] * net_out_1[i] ) ;
out_sum_2[i] = ( i_weight_4[1] * net_out_2[i] ) ;
}

for(i=0 ; i<output_var ; i++)
{
out_sum_3[i] = (1 * i_bias[2]) + ( out_sum_1[i] ) + ( out_sum_2[i] ) ;
let << "output_sum_[" << i << "]" <<"= " << out_sum_3[i] << endl ;
}

}


void final_output()
{

/////// industrial
if(n==1)
{
final_sum() ;
let<< endl<<endl ;
let<< }

for(i=0 ; i<output_var ; i++ )
{
cal_output[i][0] = ( 1.0 / ( 1.0 + exp( -out_sum_3[i] ) ) ) ;
```

```
343:
344: error[i] = ( ind_output[i][0] - cal_output[i][0] ) ;
345: let << "ind_output_[" << i << "]" << "= " << ind_output[i][0] <<
"\t\t"
346: << "final_output_[" << i << "]" << "= " << cal_output[i][0] << "\t\t"
347: << "error_vector_[" << i << "]" << "= " << error[i] << endl ;
348: }
349: }
350:
351: //// residential
352: else if(n==2)
353: {
354: final_sum() ;
355: let<< endl<<endl ;
356: let<< " \t\t\tTHE NETWORK ERROR---------> "<< endl<< endl ;
357:
358: for(i=0 ; i<output_var ; i++ )
359: {
360: cal_output[i][0] = ( 1.0 / ( 1.0 + exp( -out_sum_3[i] ) ) ) ;
361: error[i] = ( res_output[i][0] - cal_output[i][0] ) ;
362: let << "res_output_[" << i << "]" << "= " << res_output[i][0] <<
"\t\t"
363: << "final_output_[" << i << "]" << "= " << cal_output[i][0] << "\t\t"
364: << "error_vector_[" << i << "]" << "= " << error[i] << endl ;
365: }
366: }
367:
368: //////// commercial
369: else if(n==3)
370: {
371: final_sum() ;
372: let<< endl<<endl ;
373: let<< " \t\t\tTHE NETWORK ERROR---------> "<< endl<< endl ;
374:
375: for(i=0 ; i<output_var ; i++ )
376: {
377: cal_output[i][0] = ( 1.0 / ( 1.0 + exp( -out_sum_3[i] ) ) ) ;
378: error[i] = ( com_output[i][0] - cal_output[i][0] ) ;
379: let << "com_output_[" << i << "]" << "= " << com_output[i][0] <<
"\t\t"
380: << "final_output_[" << i << "]" << "= " << cal_output[i][0] << "\t\t"
381: << "error_vector_[" << i << "]" << "= " << error[i] << endl ;
382: }
383: }
384:
385:
386: }
387:
388:
389:
390:
391: void back_pass()
392: {
393: ////////////////// BACKWARDS PASS BEGINS..... ////////////////////
394: ////////// local gradients //////////
395: let<<endl << endl ;
396: for ( i=0 ; i<output_var ; i++)
397: {
398: delta_o[i] = (cal_output[i][0] * ( 1 - cal_output[i][0] ) ) ;
399: let << "delta_o_[" << i << "]" << "= " << delta_o[i] << endl ;
400: }
```

```
401:
402: let << endl << endl ;
403:
404: for ( i=0 ; i<output_var ; i++)
405: {
406: delta_h1[i] = ( net_out_1[i] * ( 1 - net_out_1[i]) ) *
407: (delta_o[i] * i_weight_4[0] ) ;
408: let << "delta_h1_[" << i << "]" << "= " << delta_h1[i] << endl ;
409: }
410:
411: let<< endl << endl ;
412:
413: for ( i=0 ; i<output_var ; i++)
414: {
415: delta_h2[i] = ( net_out_2[i] * ( 1 - net_out_2[i]) ) *
416: ( delta_o[i] * i_weight_4[1] ) ;
417: let << "delta_h2_[" << i << "]" << "= " << delta_h2[i] << endl ;
418: }
419:
420: }
421:
422:
423: void sse_error_1()
424: {
425: SSE_1=0 ;
426: squared_error_sum_1=0 ;
427:
428: for(i=0 ; i<output_var ; i++)
429: {
430: squared_error_sum_1+=pow (error[i],2) ;
431: }
432:
433: SSE_1 = (squared_error_sum_1 / 2 ) ;
434:
435: let<<endl ;
436: let << "THE SUM OF SQUARED ERROR" << endl << endl ;
437: let << "sum_squared_error_1= " << SSE_1 << endl << endl ;
438:
439: }
440:
441:
442:
443: void sse_error_2()
444: {
445: SSE_2=0 ;
446: squared_error_sum_2=0 ;
447:
448: for(i=0 ; i<output_var ; i++)
449: {
450: squared_error_sum_2+=pow (error[i],2) ;
451: }
452:
453: SSE_2 = (squared_error_sum_2 / 2 ) ;
454:
455: let << endl ;
456: let << "THE SUM OF SQUARED ERROR" << endl << endl ;
457: let << "sum_squared_error_2= " << SSE_2 << endl << endl ;
458:
459: }
460:
461:
```

```
462:
463:
464: ////////////////////////////// forward and backward computation
//////////
465:
466: void FB_computation_1()
467: {
468: final_output() ; back_pass() ; sse_error_1() ;
469: }
470:
471:
472: void FB_computation_2()
473: {
474: final_output() ; back_pass() ; sse_error_2() ;
475: }
476:
477:
478:
////////////////////////////////////////////////////////////////////////
////
479:
////////////////////////////////////////////////////////////////////////
////
480: ////////////////////// jacobian matrix all rows
//////////////////////////////
481:
482: void jacobian_matrix()
483: {
484: FB_computation_1() ;
485:
486: ////////////// if industrial inputs/outputs variable is selected for
training ///
487: if(i==1)
488: {
489: //////// hidden layer 1 jacobian ///////
490:
491: for(i=0 ; i<output_var ; i++)
492: {
493: for(j=0 ; j<input_var ; j++)
494: {
495: H_layer1[i][j] = (ind_input[i][j] * delta_h1[i] ) ;
496: }
497: }
498:
499: ////// hidden layer 2 jacobian ///////////////
500:
501: for( i=0 ; i<output_var ; i++ )
502: {
503: for(j=0 ; j<input_var ; j++)
504: {
505: H_layer2[i][j] = ( ind_input[i][j] * delta_h2[i] ) ;
506: }
507: }
508:
509: ////// output layer jacobian ////////
510: for( i=0 ; i<output_var ; i++ )
511: {
512: o_layer_1[i] = ( net_out_1[i] * delta_o[i] ) ;
513: o_layer_2[i] = ( net_out_2[i] * delta_o[i] ) ;
514: }
515:
```

```
516: ////// hidden layer 1 bias jacobian ////////
517: for(i=0 ; i<output_var ; i++)
518: {
519: H_layer_3[i] = ( cal_output[i][0] * delta_h1[i] ) ;
520: H_layer_4[i] = ( cal_output[i][0] * delta_h2[i] ) ;
521: }
522:
523:
524: ////// hidden layer 1 bias jacobian ////////
525: for(i=0 ; i<output_var ; i++)
526: {
527: j_bias1[i] = ( i_bias[0] * delta_h1[i] ) ;
528: }
529:
530: ////// " hidden layer 2 bias jacobian " ///////////
531: for(i=0 ; i<output_var ; i++)
532: {
533: j_bias2[i] = ( i_bias[1] * delta_h2[i] ) ;
534: }
535:
536:
537: ///////" output layer bias jacobian " ///////////
538: for(i=0 ; i<output_var ; i++)
539: {
540: j_bias3[i] = ( i_bias[2] * delta_o[i] ) ;
541: }
542:
543: }
544:
545:
////////////////////////////////////////////////////////////////////////////
////
546:
////////////////////////////////////////////////////////////////////////////
////
547:
548: //////////// if residential input/output is selected for training
//////////////
549: if(i==2)
550: {
551: ///////// hidden layer 1 jacobian ////////
552:
553: for(i=0 ; i<output_var ; i++)
554: {
555: for(j=0 ; j<input_var ; j++)
556: {
557: H_layer1[i][j] = (res_input[i][j] * delta_h1[i] ) ;
558: }
559: }
560:
561: ////// hidden layer 2 jacobian ///////////////
562:
563: for( i=0 ; i<output_var ; i++ )
564: {
565: for(j=0 ; j<input_var ; j++)
566: {
567: H_layer2[i][j] = ( res_input[i][j] * delta_h2[i] ) ;
568: }
569: }
570:
571: ////// output layer jacobian /////////
```

```
572: for( i=0 ; i<output_var ; i++ )
573: {
574: o_layer_1[i] = ( net_out_1[i] * delta_o[i] ) ;
575: o_layer_2[i] = ( net_out_2[i] * delta_o[i] ) ;
576: }
577:
578: ////// hidden layer 1 bias jacobian ////////
579: for(i=0 ; i<output_var ; i++)
580: {
581: H_layer_3[i] = ( cal_output[i][0] * delta_h1[i] ) ;
582: H_layer_4[i] = ( cal_output[i][0] * delta_h2[i] ) ;
583: }
584:
585:
586: ////// hidden layer 1 bias jacobian ////////
587: for(i=0 ; i<output_var ; i++)
588: {
589: j_bias1[i] = ( i_bias[0] * delta_h1[i] ) ;
590: }
591:
592: ////// " hidden layer 2 bias jacobian " ///////////
593: for(i=0 ; i<output_var ; i++)
594: {
595: j_bias2[i] = ( i_bias[1] * delta_h2[i] ) ;
596: }
597:
598:
599: ///////" output layer bias jacobian " ///////////
600: for(i=0 ; i<output_var ; i++)
601: {
602: j_bias3[i] = ( i_bias[2] * delta_o[i] ) ;
603: }
604:
605: }
606:
607: ////////////////////// jacobian matrix row 1
//////////////////////////////
608:
609: for(i=0 ; i<(output_var) ; i++)
610: {
611: for(j=0 ; j< ((2*input_var) + (2*hidden_neuron) +
612: (hidden_neuron + output_neuron ) ) ; j++ )
613: {
614:
615: if ( j>=0 && j<6 )
616: {
617: jac[i][j]= H_layer1[i][j] ;
618: }
619:
620: if ( j>=6 && j<12 )
621: {
622: jac[i][j]= H_layer2[i][j-6] ;
623: }
624:
625:
626: if( j==12 )
627: {
628: jac[i][j]= H_layer_3[j-12+i] ;
629: }
630:
631: if( j==13 )
```

```
632: {
633: jac[i][j]= H_layer_4[j-13+i] ;
634: }
635:
636:
637: if( j==14 )
638: {
639: jac[i][j]= o_layer_1[j-14+i] ;
640: }
641:
642: if( j==15 )
643: {
644: jac[i][j]= o_layer_2[j-15+i] ;
645: }
646:
647: if( j==16 )
648: {
649: jac[i][j]= j_bias1[j-16+i] ;
650: }
651:
652: if( j==17 )
653: {
654: jac[i][j]= j_bias2[j-17+i] ;
655: }
656:
657: if( j==18 )
658: {
659: jac[i][j]= j_bias3[j-18+i] ;
660: }
661: }
662: }
663: }
664:
////////////////////////////////////////////////////////////////////////////
665:
////////////////////////////////////////////////////////////////////////////
666:
667: /////////// if commercial input/output is selected for training
///////////
668:
669: if(i==3)
670: {
671: ///////// hidden layer 1 jacobian ////////
672:
673: for(i=0 ; i<output_var ; i++)
674: {
675: for(j=0 ; j<input_var ; j++)
676: {
677: H_layer1[i][j] = (com_input[i][j] * delta_h1[i] ) ;
678: }
679: }
680:
681: ////// hidden layer 2 jacobian ///////////////
682:
683: for( i=0 ; i<output_var ; i++ )
684: {
685: for(j=0 ; j<input_var ; j++)
686: {
687: H_layer2[i][j] = ( com_input[i][j] * delta_h2[i] ) ;
688: }
689: }
```

```
690:
691: ////// output layer jacobian /////////
692: for( i=0 ; i<output_var ; i++ )
693: {
694: o_layer_1[i] = ( net_out_1[i] * delta_o[i] ) ;
695: o_layer_2[i] = ( net_out_2[i] * delta_o[i] ) ;
696: }
697:
698: ////// hidden layer 1 bias jacobian ////////
699: for(i=0 ; i<output_var ; i++)
700: {
701: H_layer_3[i] = ( cal_output[i][0] * delta_h1[i] ) ;
702: H_layer_4[i] = ( cal_output[i][0] * delta_h2[i] ) ;
703: }
704:
705:
706: ////// hidden layer 1 bias jacobian ////////
707: for(i=0 ; i<output_var ; i++)
708: {
709: j_bias1[i] = ( i_bias[0] * delta_h1[i] ) ;
710: }
711:
712: ////// " hidden layer 2 bias jacobian " //////////
713: for(i=0 ; i<output_var ; i++)
714: {
715: j_bias2[i] = ( i_bias[1] * delta_h2[i] ) ;
716: }
717:
718:
719: ///////" output layer bias jacobian " //////////
720: for(i=0 ; i<output_var ; i++)
721: {
722: j_bias3[i] = ( i_bias[2] * delta_o[i] ) ;
723: }
724:
725: }
726:
727: if(i==1)
728: {
729: jacobian << " \t\tRESIDENTIAL JACOBIAN MATRICES " << endl << endl ;
730:
731: {
732: for(j=0 ; j< ((2*input_var) + (2*hidden_neuron) +
733: (hidden_neuron + output_neuron ) ) ; j++ )
734: {
735: jacobian << jac[i][j] << " \t\t" ;
736: }
737: jacobian << endl ;
738: }
739: }
740:
741: if(i==2)
742: {
743: jacobian << " \t\tRESIDENTIAL JACOBIAN MATRICES " <<endl << endl ;
744: for(i=0 ; i<(output_var) ; i++)
745: {
746: for(j=0 ; j< ((2*input_var) + (2*hidden_neuron) +
747: (hidden_neuron + output_neuron ) ) ; j++ )
748: {
749: jacobian << jac[i][j] << " \t\t" ;
750: }
```

```
751: jacobian << endl ;
752: }
753: }
754:
755: if(i==3)
756: {
757: jacobian << " \t\tCOMMERCIAL JACOBIAN MATRICES " << endl << endl ;
758: for(i=0 ; i<(output_var) ; i++)
759: {
760: for(j=0 ; j< ((2*input_var) + (2*hidden_neuron) +
761: (hidden_neuron + output_neuron ) ) ; j++ )
762: {
763: jacobian << jac[i][j] << " \t\t" ;
764: }
765: jacobian << endl ;
766: }
767: }
768: }
769:
770:
771:
772:
773: int main(int argc, char *argv[])
774: {
775: cout << "NUMBER OF INPUT VARIABLE = " ;
776: cin >> input_var ;
777: cout << endl ;
778:
779: cout << "NUMBER OF OUTPUT VARIABLE = " ;
780: cin >> output_var ;
781: cout << endl ;
782:
783: cout << "NUMBER OF HIDDEN NEURON= " ;
784: cin >> hidden_neuron ;
785: cout << endl ;
786:
787: cout << "NUMBER OF OUTPUT NEURON= " ;
788: cin >> output_neuron ;
789: cout << endl ;
790:
791:
792: cout << "Enter : 1 ------> for industrial input : output " << endl ;
793: cout << "Enter : 2 ------> for residential input : output " << endl ;
794: cout << "Enter : 3 ------> for commercial input : output " << endl ;
795:
796: cin >> n ; //// ask for input
797:
798: switch(n)
799: {
800: case 1 : {
801: cout<< "you have selected < INDUSTRIAL INPUT : OUTPUT >" << endl <<
endl ;
802:
803: }
804: break ;
805:
806: case 2 : {
807: cout<< "you have selected < RESIDENTIAL INPUT : OUTPUT > " << endl
<<endl ;
808:
809: }
```

```
810: break ;
811:
812: case 3 : {
813: cout<< "you have selected < COMMERCIAL INPUT : OUTPUT > " << endl <<
endl ;
814:
815: }
816: break ;
817: }
818:
819:
820:
821: jacobian_matrix() ;
822:
823: system("PAUSE");
824: return EXIT_SUCCESS;
825: }
826:
827:
828:
```