ASSOCIATION POUR LE DÉVELOPPEMENT DE L'ENSEIGNEMENT ET DE LA RECHERCHE EN SYSTÉMATIQUE APPLIQUÉE

MELISSA

Study for the non linear Model Based Predictive Control of Spirulina compartment using knowledge model

Contract ESA-ESTEC / ADERSA
Purchase order n° 142356 of 30/06/94

Memorandum of Understanding
ECT/FG/CB/95.205

Technical Note 24.2
Version 1 - Issue 1

N. FULGET

- October 1995 -

7, bd du Maréchal Juin
B.P. 52
91371 VERRIERES-LE-BUISSON Cedex
Téléphone : (1) 60 13 53 53
Télécopie : (1) 69 20 05 63

ADERSA

Document change log

Issue	Date	Observations
00	08/95	Draft
01	09/95	Corrected version (corrections and remarks of the 15.09.95)

SUMMARY

INTRODUCTION

In the previous technical notes, a first approach of Model Based Predictive Control, with linear model has been presented (TN 21.1), and a Simulink Simulator of the Spirulina compartment has been elaborated (TN 21.2). This simulator is based on the first principles model, developped in LGCB (TN 19.1, 19.2, 19.3) which is a non linear dynamical model. In TN 24.1, the non linear knowledge model has been validated. The experimental results have been compared to simulation results for different dilution rates, and for different steps of light. After, a non linear model based predictive control law has been developped and tested in simulation, on the non linear simulator (TN 24.1).

The aim of this study is to present the experimental results obtained on the real spirulina photosynthetic reactor, with the non linear control law presented in TN 24.1.

In this technical note, the integration and implementation of the control law is mentioned, and then different experimental results are presented and discussed.

II - IMPLEMENTATION OF THE PREDICTIVE CONTROL LAW IN THE GPS SOFTWARE

II.1. Control strategy

The control strategy and methodology have been presented in details in TN 24.1. It concerns only the control of the biomass production in Spirulina compartment by action on the light intensity. We can remind the main principles. It can be decomposed in 3 levels (hierarchical structure).

Level 0 : It consists in the regulation of the light intensity. The measure is the light intensity in the center of the reactor E_b. Level 0 calculates the power to apply to the lamps to regulate the light intensity in the center of the reactor. This action is calculated with a classical PI controller.

Level 1 : It consists in the regulation of biomass production by action on the light intensity. The control law is a non linear predictive control law, which uses the non linear knowledge model and which consists in applying on it some scenarios of radiant flux values F_r during the prediction horizon (see TN 24.1).

The available model allows to calculate a radiant flux F_r, which is converted in a corresponding value E_b, of light intensity in the center of the reactor, through a non linear model, using the measure of the biomass concentration.

Level 2 : The role of this level is the optimisation of setpoints, with respect to constraints. It calculates feasible production and flow setpoints in order to respect the constraints and to optimize the functioning.

The hierarchical structure is presented on figure 1.

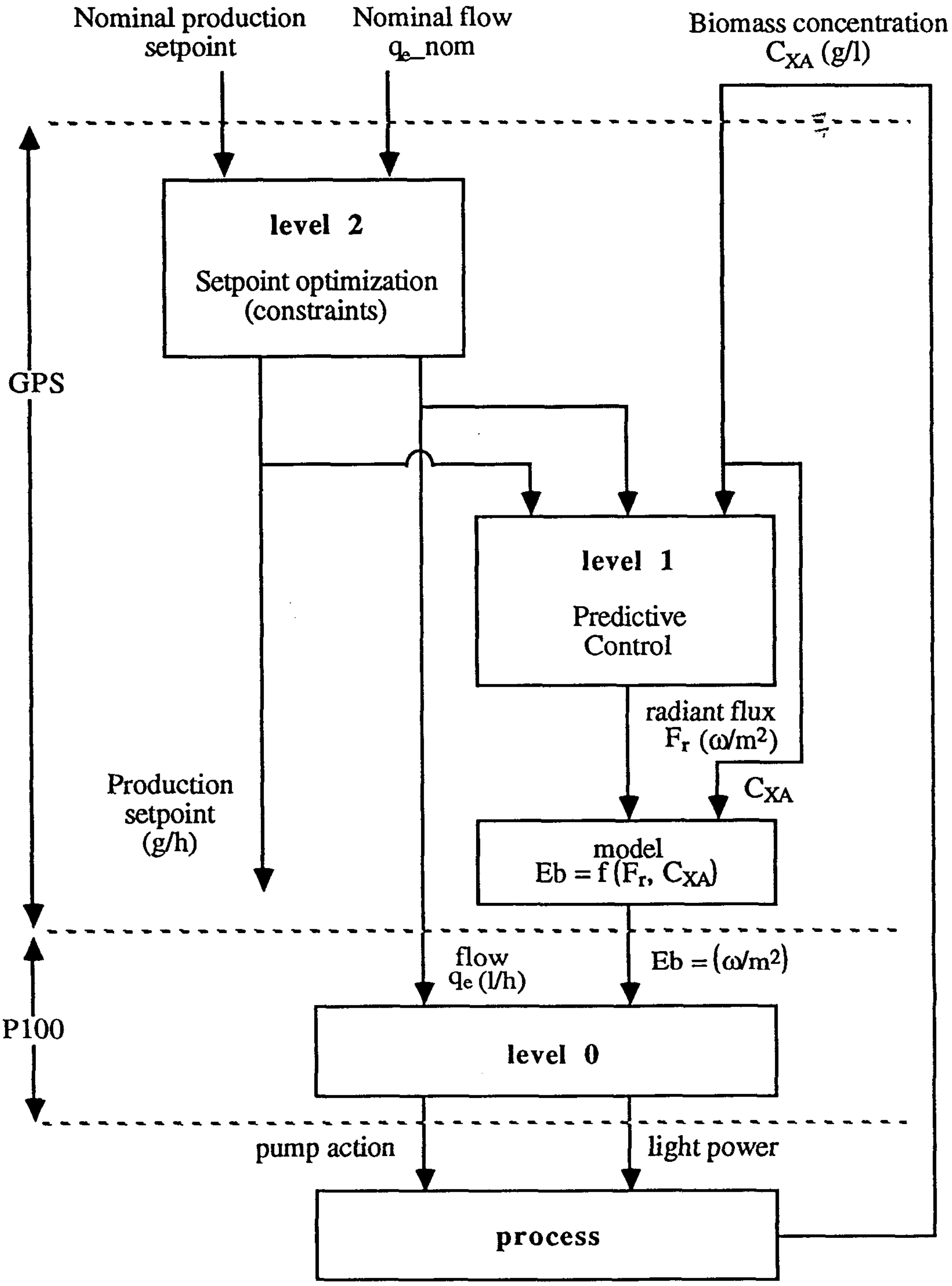


Figure 1 - Hierarchical structure of the control law

II.2. Implementation of the control law

In TN 24.1, a control law (non linear predictive) has been presented and tested in simulation. For the simulations, the software MATLAB Simulink has been used. As it offers the possibility to interface C code with MATLAB Simulink routines, it has been decided to write directly the control software in C code. This C code program is given in annex B of TN 24.1 and in annex A of this present technical note. This procedure facilitates the real time implementation, but this C code had to be integrated in the specific GPS station. This has been done with the assistance of C. Binois, from MATRA, who has previously realized the control system of Spirulina compartment, on the GPS station ([1] and [2]).

The description of the GPS station, and of the control system architecture is presented in TN 18.3. The control software was implemented on the GPS station, in C langage. This software reads the values given by the sensors, and by the user station, through the network. It calculates different statistical values (average, slope, variance ...), and the action values which are transmitted through the network to the P100 controllers, which pilot the process (figure 2).

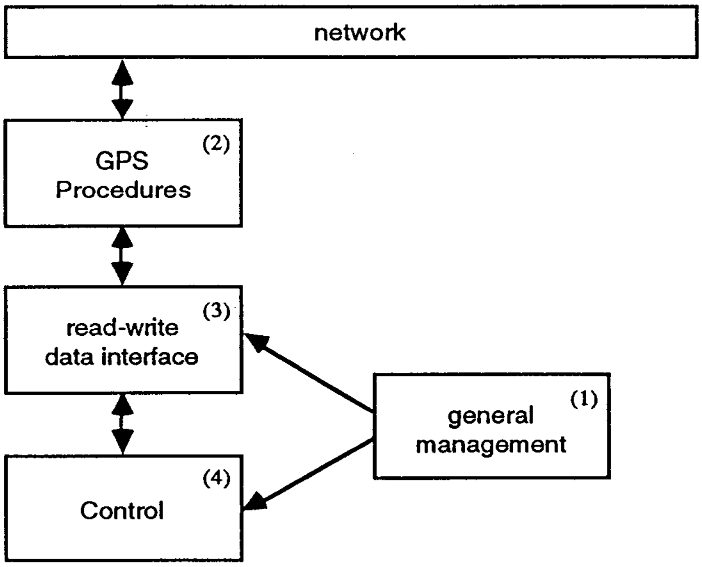


Figure 2 - Control system architecture

II.3. Description of the control law software

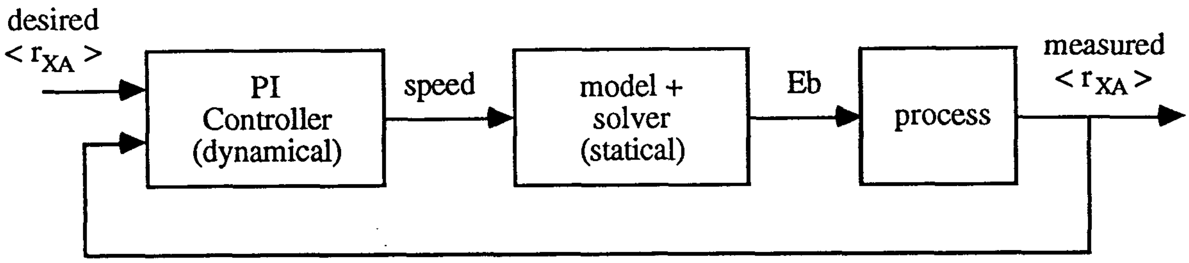
We only focus on the control program, that is, the part (4) of the control system architecture. The file control.c (annex B), written by C. Binois, and which corresponds to the part (4) of the control system architecture will be replaced by a program containing the non linear predictive control law. The control program given in annex B contains a first version of control law, developped by C. Binois and presented in his thesis report [2]. This control law (based on a PI strategy and a non linear model inversion) has been replaced by the non linear predictive control law, developped by ADERSA, which is a more powerful strategy.

- Main differences between the strategy developped by BINOIS and the strategy proposed by ADERSA

  - In the control strategy developped by BINOIS ([1] and [2]), the biomass concentration was maintained constant, at a certain value, by action of the pump flow.

  - The knowledge model was used in a statical way $\left(\frac{dC_{XA}}{dt} \cong 0\right)$. This model was inverted by a solver to find the value of Eb which would give a desired production speed.

  - The dynamical closed loop was realized with a linear PI controller which calculated the production speed in function of the difference between the desired growth rate and the measured growth rate.



"BINOIS" strategy (PI + model)

  - In the new strategy, because of the functioning constraints of the complete loop, the pump flow is supposed to be fixed at a certain value. So, the concentration can't be maintained at a fixed value. Then, the process can be at different functioning points, and its behaviour is more non linear. So, we have proposed a more powerful control strategy where the non linear model is used in the dynamic closed loop, in a non linear predictive control law. All the more, the strategy is more simpler to tune.

- P100 controllers

  In the P100, several controllers are implemented (pH control, level control, biomass control, light control ...). Those controllers are basic controllers (PID).

  - The light controller calculates the power of the lamps to control the light intensity in the center of the reactor.

  - The biomass controller is not in closed loop in the present strategy. Indeed, in the previous strategy, the biomass concentration was controlled with action of pump flow, but in the present strategy, the value of the flow is given and the biomass concentration is free of control. So the desired pump flow value is put in the place of the closed loop calculated value.

- Last version (PI controller + solver)

  The file *control.c,* previously written by C. Binois (annex B) was containing a routine *model (REACT * react).* It corresponds to the mathematical model of Spirulina growth. This model has two functional inputs : the biomass concentration C_{XA} and the light intensity in the center of the reactor E_b. With those two inputs (which correspond to measurements on the process), a value of radiant flux $F_r$and a value of mean growth rate $<r_{XA}>$ can be determined.

  The file *control.c* was also containing a routine *solver (REACT * react, double Rx_seek)* which calculated the value of light intensity E_b which would give a desired mean growth rate. It corresponds to the inversion of the previous non linear model.

  The desired mean growth rate was determined by a classical PI controller, contained in the routine *control_spiru ( ).*

- New version (non linear predictive control)

  In order to integrate the new control law, some modifications have been done in the file *control.c.* The final result is given in annex C. It can be noted that the routine *model (REACT * react, int mode)* has an additional argument *mode* which allows to calculate either

  - the radiant flux F_r in function of the biomass concentration C_{XA}, and the light intensity in the center of the reactor E_b, or

  - the light intensity E_b, in function of the biomass concentration C_{XA} and the radiant flux F_r. The model can be used in both case. Indeed, the two relations $E_b = f(F_r, C_{XA})$ and $F_r = f(E_b, C_{XA})$ are known. In the predictive control strategy, the input is supposed to be F_r. In the control terminilogy, F_r is named the manipulated variable. We have chosen this solution because the domain of F_r variations is fixed, constant and known, even though the domain of E_b variations is depending on the concentration value. So, it is easier to determine the value of the scenarios with F_r, than with E_b.

The predictive control strategy is programmed in the routine *control_spiru ( ).* In fact, the level 0 is contained in the P100 controller, and the level 1 and 2 are contained in the GPS station. The routine *solver (REACT * react, double Rx_seek)* is suppressed. The solver is in fact contained in the predictive strategy (scenario). The routine *model (REACT * react, int mode)* gives just the calculation of the mean growth rate $<r_{XA}>$, but the output of the model used in the predictive strategy is the biomass production. Its functional inputs are the radiant flux F_r and the dilution rate *dil* (*dil = flow/volume*). This model, named internal model is contained in the routine *adersa (double flux, double dil1),* which calls the routine *model (REACT * react, int mode)* to calculate the mean growth rate $<r_{XA}>$. The routine *adersa* integrates the following differential equation :

$$\frac{dC_{XA}}{dt} = - \text{dil} \, . \, C_{XA} + \langle r_{XA} \rangle$$

on the prediction horizon N_{HC}, that is from the current time n, to the prediction time $n + N_{HC}$. The dilution rate and the radiant flux are supposed to be constant on the prediction horizon. The biomass concentration C_{XA} is supposed to be initialized to the measured value, at current time. The differential equation is integrated with Euler method, with a constant step equal to the sampling period (1/2 hour).

The production is directly the product of the dilution rate, with the concentration and the volume. It is expressed in mg/h.

Description of the routine *control_spiru ( )* : this routine is called each minute, however all items regarding level 2 and 1 are called only once per 30 minutes.

At the beginning of the routine, the acquisition of measured or estimated values is done. The production *prod* is calculated with the average of biomass concentration measurements on the 10 last minutes, *cxa_moy*, and the measure of the flow *qe_real* :

prod = cxa_moy * qe_real

After, maximal and minimal values of flow and production are expressed for the level 2. The flow constraints *qe_max* and *qe_min* are calculated with the nominal flow value *qe_nom*, given by the operator, and with a given maximal variation around this nominal flow equal to *DQ* :

qe_max = qe_nom * (1 + DQ)
qe_min = qe_nom * (1 - DQ)

The maximal production is the product of the maximal concentration with the maximal flow. The minimal production is the product of the minimal concentration with the minimal flow :

prod_max = CXA_MAX * qe_max
prod_min = CXA_MIN * qe_min

Maximal concentration *CXA_MAX*, minimal concentration *CXA_MIN*, and maximal variation of flow *DQ* are parameters. In this version, there values are :

DQ = 0.1	(0 < DQ < 1)
CXA_MAX = 1.5	(g/l)
CXA_MIN = 0.5	(g/l)

but they can be modified if necessary. They are contained in file MELISSA.H.

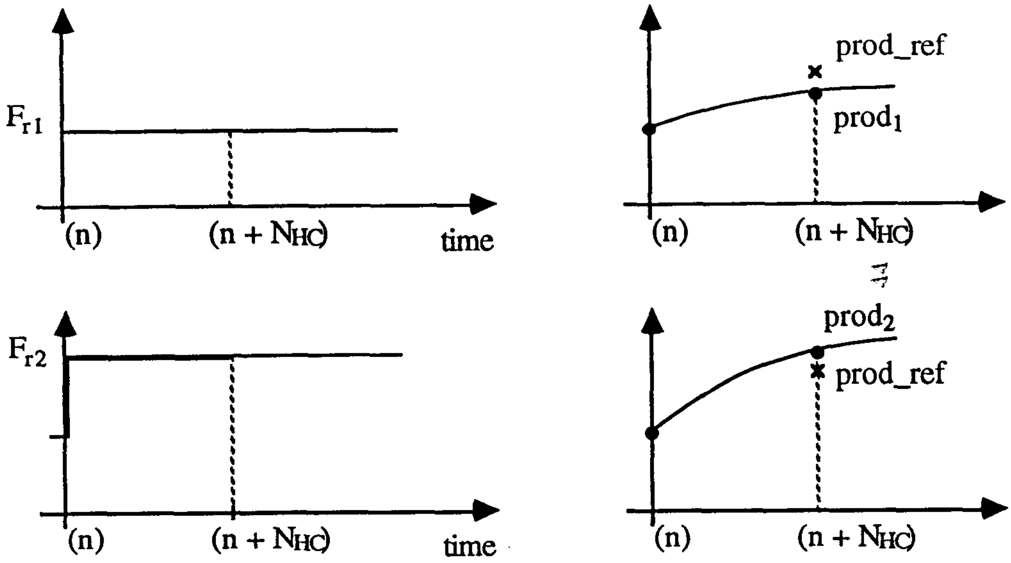
Then level 2 calculates a feasible production setpoint *cons_prod_real* with respect to constraints on the production, and a feasible flow setpoint *qe_real* with respect to constraints, in order to optimize, if necessary, the production. The dilution rate *dil* is then deduced from this new value of flow. The feasible production setpoint *cons_prod_real* and the dilution rate *dil* is transmitted to level 1 in order to determine, with a non linear predictive control strategy (scenario strategy), a value of radiant flux setpoint F_r.

The first step of level 1 is the determination of reference trajectory *prod_ref,* in function of the measured production *prod,* and of the feasible production setpoint *cons_prod_real.* The reference trajectory is a first order function (exponential). Its response time (at 95%) TR_{95} can be chosen. It allows to fixe the closed loop time response. In fact, the tuning parameter is LAMBDA (the logarithmic decrement) :

$$\text{LAMBDA} = \exp\left(-\frac{3\ \text{DT}}{\text{TR}_{95}}\right)$$

where DT is the sampling period. Reference trajectory corresponds to the control objective in the future.

The second step of level 1 is the determination of radiant flux value in order to satisfy this control objective in the future. As the model is non linear, the strategy used to determine the radiant flux value is a scenario strategy (see TN 24.1). It consists in applying different values of radiant flux to the model, during the prediction horizon. In this case, it is sufficient to apply 2 scenarios. The first scenario is applied with a radiant flux F_{r1} equal to the real radiant flux applied on the process. The second scenario F_{r2} is calculated with an increase or a decrease (in function of the sign of the difference between setpoint and measured production). The value of the increase is a parameter. In this version, it is equal to 10 W/m^2. With the first scenario applied on the model *adersa,* the model production obtained is equal to *prod1.* With the second scenario, the model production obtained is equal to *prod2.* Then the setpoint for radiant flux is determined in function of F_{r1}, F_{r2}, *prod1*, *prod2* and the desired production *prod_ref,* supposing that the relation between F_r and production is linear. The obtained radiant flux value F_r is passed through the minimal and maximal constraints (FR_MIN = 10 W/m^2, FR_MAX = 400 W/m^2),.



$$F_r = F_{r1} + \frac{(F_{r2} - F_{r1})(\text{prod\_ref} - \text{prod1})}{\text{prod}_2 - \text{prod}_1}$$

$$F_r = \max(\text{FR\_MIN}, \min(\text{FR\_MAX}, F_r))$$

Then the corresponding setpoint for the light intensity in the center of the reactor E_b, is calculated in function of the constrained radiant flux setpoint F_r, and the biomass concentration average. The setpoint for the pump corresponding to the desired flow *qe_real*, is also calculated in function of conversion parameters.

$$\text{Eb} = f(F_r, C_{XA}\text{\_moy})$$

$$\text{act\_pompe} = \text{qe\_real} * \frac{1000}{60 \,.\, \text{cpt\_cxa\_unit}} \qquad \left(0 < \text{act\_pompe} < 1\right)$$

cpt_cxa_unit is a caracterisation parameter of the pump, calculated by BINOIS according to the following property (0 < act_pompe < 1) : when the maximal flow is applied during one minute, the value of the counter is equal to *cpt_cxa_unit*.

The setpoint for the light intensity E_b in the reactor, and the setpoint for the pump action are sended through the network to the P100 controllers, which act directly on the process, through actuators. The value of *act_pompe* is put in the LOC_0154, and the value of E_b is put in the LOOP_0105.

III - EXPERIMENTAL RESULTS

III.1. Introduction

All the tests presented here have been done by C. LASSEUR, from ESTEC, on the Spirulina compartment, in ESTEC, before the removal to Barcelona.

First tests have been realized in order to validate the programming. They allowed to solve some writting problems and errors.

When the programming has seemed to be correct, other tests have been done to test the control methodology at different values of production setpoint.

III.2. Results description

Results of tests are presented day per day in annex E. The represented variables concern the production, the flow, the biomass concentration and the light intensity. They are plotted on four graphs.The time is graduated in hours.

On the first graph, all the variables are concerning the biomass production, in mg/h. They are listed here after :

- the production setpoint (continuous line)
- the production reference (internal variable of the predictive control method) (dotted ...)
- the measured production (dashed - -).

On the second graph, the variables are concerning the flow, in l/h. They are listed here after:

- the nominal setpoint of flow (continuous line)
- the real flow (dashed - -).

The real flow can be different from the nominal setpoint because of the strategy developped in level 2 of control, in order to optimize the production and to respect the constraints.

On the third graph, the biomass concentration is represented in mg/l. The biomass concentration is measured each minute, but the measure that is really used is a sliding average on the 10 last minutes.

On the fourth graph, the measure of light intensity in the center of the reactor, and its setpoint are represented in W/m^2. When they are not equal, the measure is the saturated one. Peaks are only on the measure.

In annex E, all the tests done from 8 of March to 13 of April are represented. In order to be explained and discussed, some of them have been selected, and are presented here after on figures 3 to 10.

On figure 3 (from 09/03 to 13/03), the setpoints of production and flow have been maintained constant a long time. During that time, the real production is constant but not equal to its setpoint. There is a steady state error. The setpoint is equal to 80 mg/h and the real production is equal to 70 mg/h. Even when a step of production setpoint is realized (from 80 mg/h to 100 mg/h) , the real production, the concentration and the light intensity don't vary. They should have increased roughly when the setpoint has changed. On figure 4 (from 14/03 to 16/03, the behaviour was not correct. The setpoint was not reached by the system. After those tests, it was concluded that there were some problems in the control software implementation. The problem was due to a division of *DT* by 60 (DT is an integer, 60 was considered as an integer. So, the division was done in the integer domain. And the result was always equal to zero. So, we have replaced <60> by <60.>. After this correction, another test was realized from 18/03 to 19/03 (figure 5). This test has consisted in an increase of production setpoint from 60 mg/h to 80 mg/h. The light E_b,has then immediatly increased. As the flow is maintained constant, the concentration has increased and then the production has increased to the new setpoint value.

The time response, at 95% is nearly 15 hours. The static precision of production control is smaller than 5%. The light action is smooth. The constraint is respected. So, the level 0 of E_b control can follow its setpoint (E_b_setpoint ≈ E_b_measure). The peaks are just on the measure. Those are not physical but measurement peaks.

Another interesting test has been realized from 29/03 to 2/04 (figure 6) with 3 steps of production setpoint. The two first steps have a good response without any modifications in the flow. The time response, at 95% is nearly 15 hours. The static precision is smaller than 5%. But for the third step, as the concentration is on its high constraint, the flow has increased of 10 %. The increase of production is realized with an increase of flow, and not with an increase of light.

For the first step, it can be noticed that the setpoint E_b can not be realized. This is due to a false value of maximal constraint on F_r. In Melissa.h, *FR_MAX* was equal to 8000 W/m^2. It should have been equal to 400 W/m^2.

After, a decreasing step of production is realized from 3/04 to 9/04 (figures 7 and 8). During all this decreasing phase, the light is on its low constraint. The only way to have a decrease of production is the dilution phenomenon. The decrease is very long and very slow. When the production setpoint is set to 50 mg/h, the real flow should have been set to the nominal value. The level 2 programming was not correct. The calculation of qe_real was not taken into account correctly. It was not refreshed when the constraints were no more reached. In order to test this problem, a great variation of production setpoint has been applied to the system the 12/04 (figure 9). A last test, done the 13/04, with an increase of production setpoint from 65 mg/h to 90 mg/h has given good results, with a time response equal to 12 hours. It can be remarked that the measure of E_b is saturated to 160 W/m^2, which is a very high value.

III.3.Conclusion

Those tests have allowed to validate and to correct the control software. They also have allowed to validate the control methodology in a certain domain of functionning : low dilution rate. All the tests have been done,with the same value of dilution rate (dil = 0.01 h^{-1}). So, some complementary tests will have to be done, at high dilution rate (dil ≈ 0.03 h^{-1}), and under minimal limitations (nitrate concentration under Monod constant value).

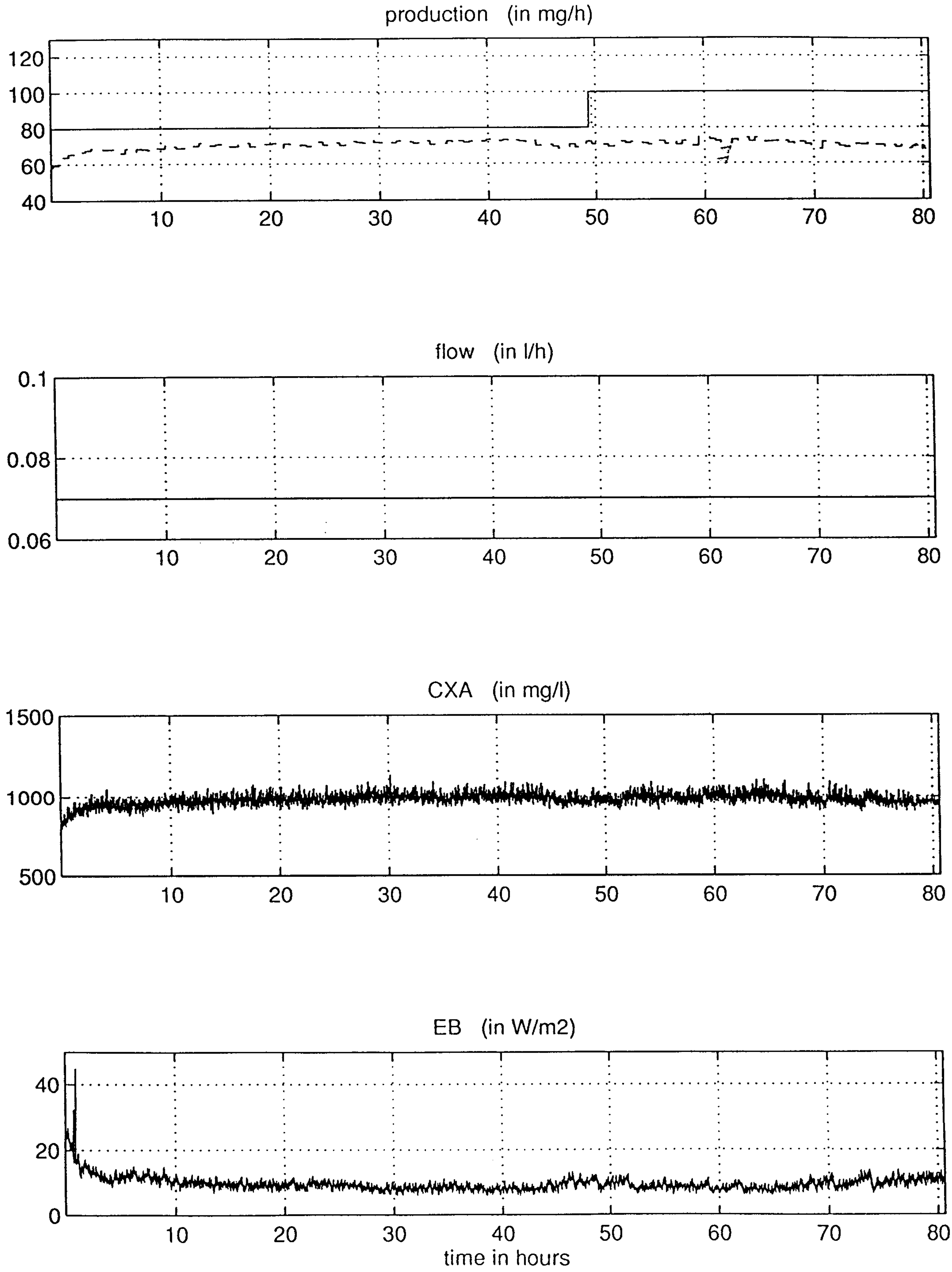


figure 3: experimental results from 09/03 to 13/03

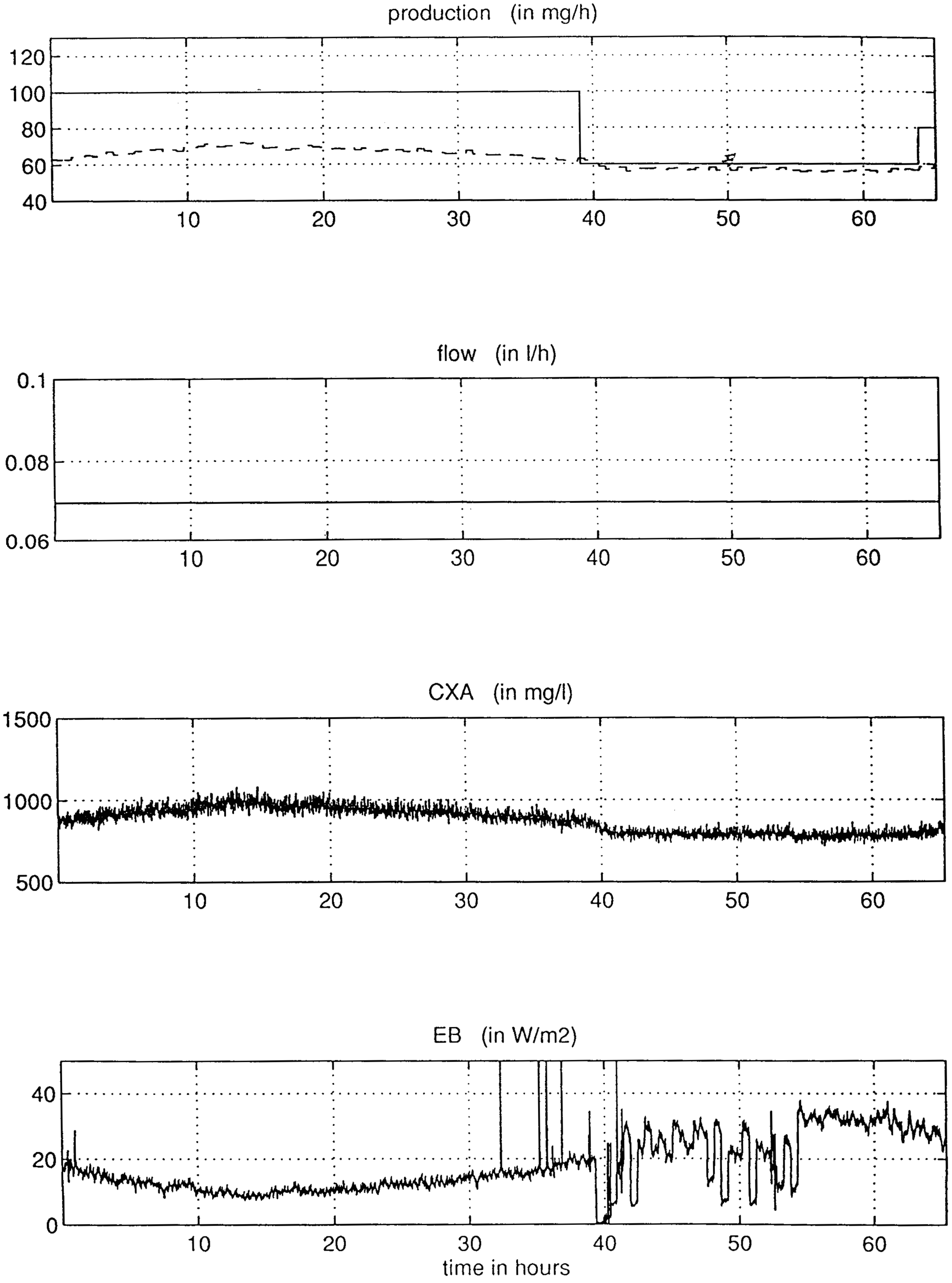


figure 4: experimental results from 14/03 to 16/03

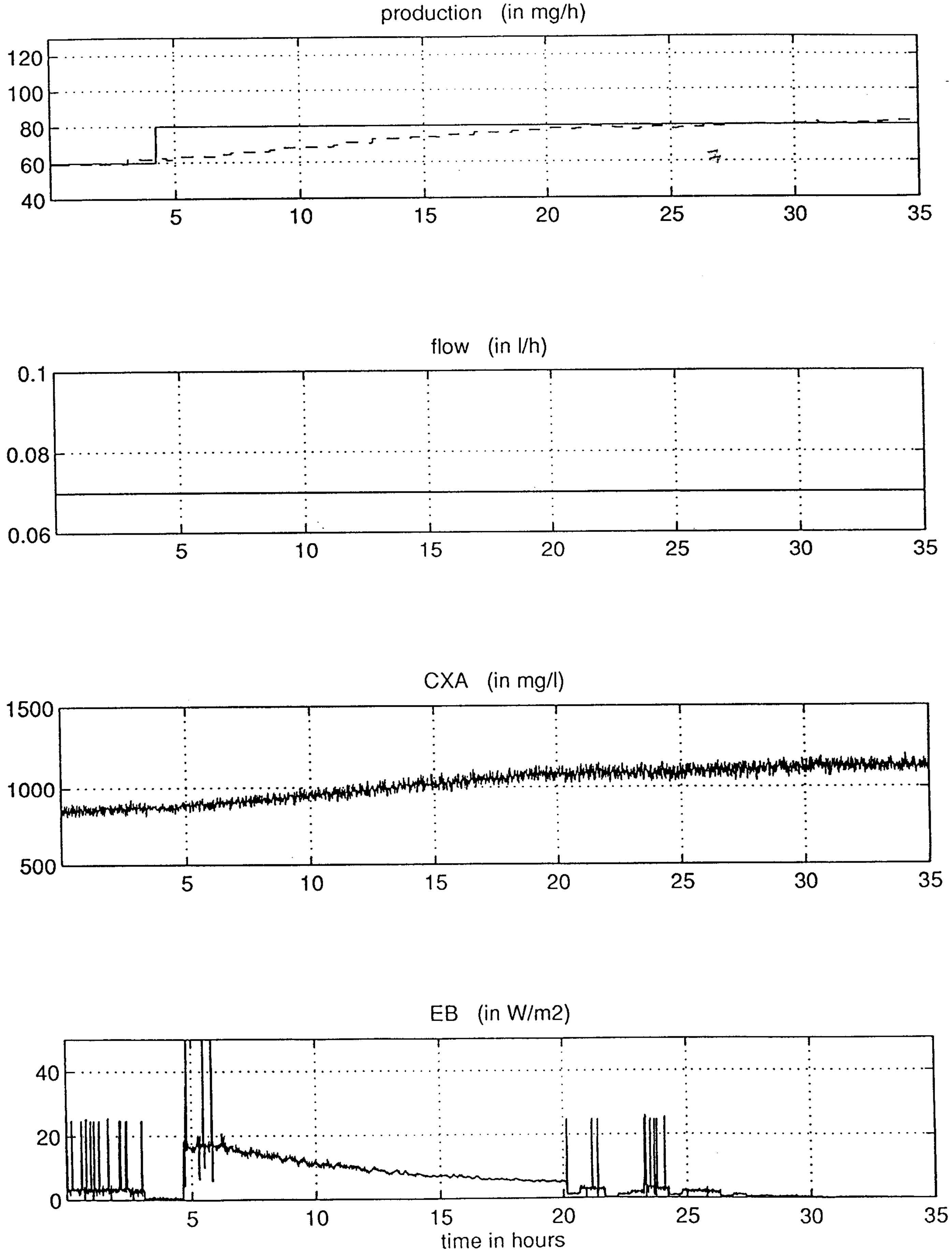


figure 5: experimental results from 18/03 to 19/03

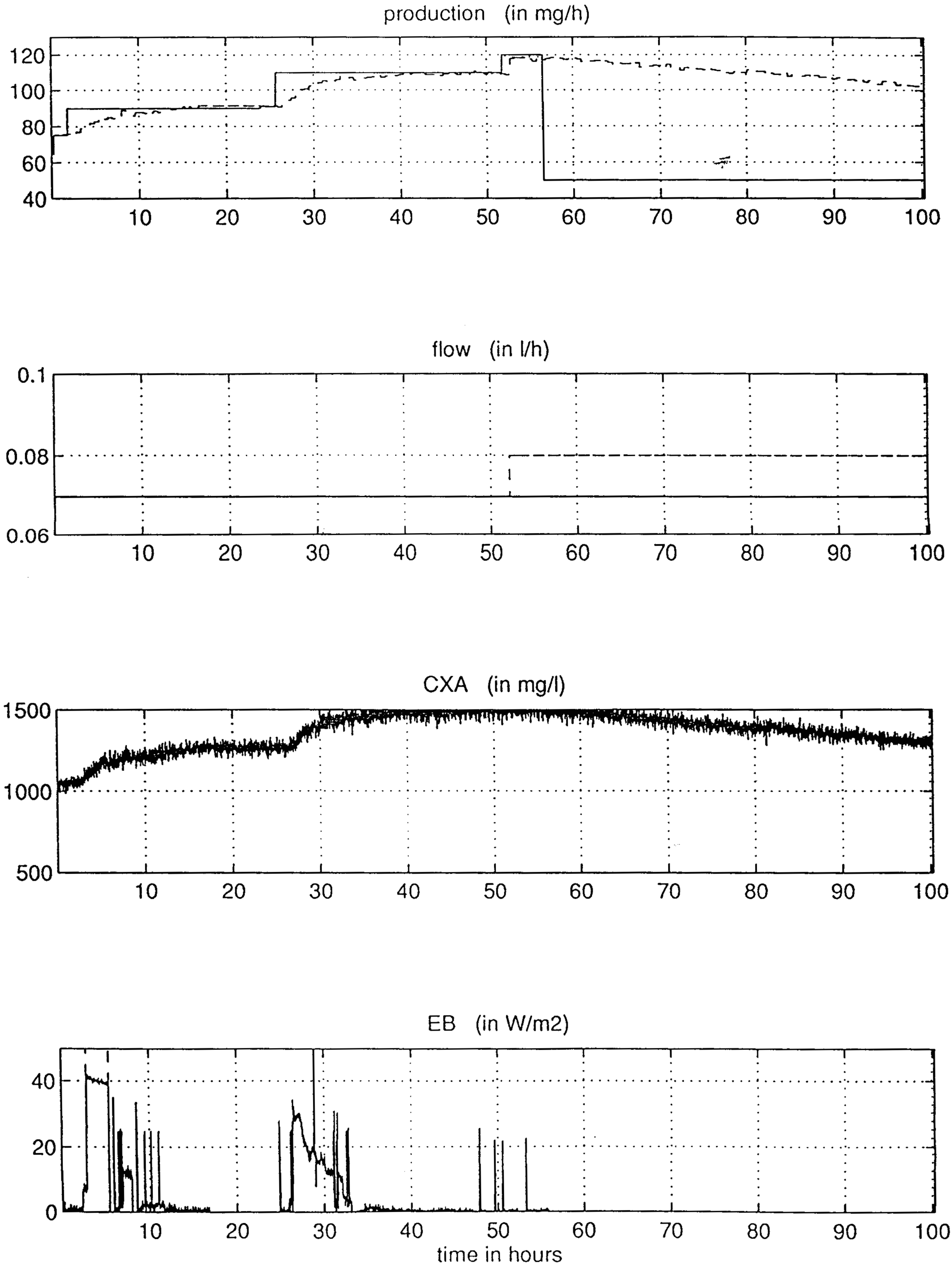


figure 6: experimental results from 29/03 to 02/04

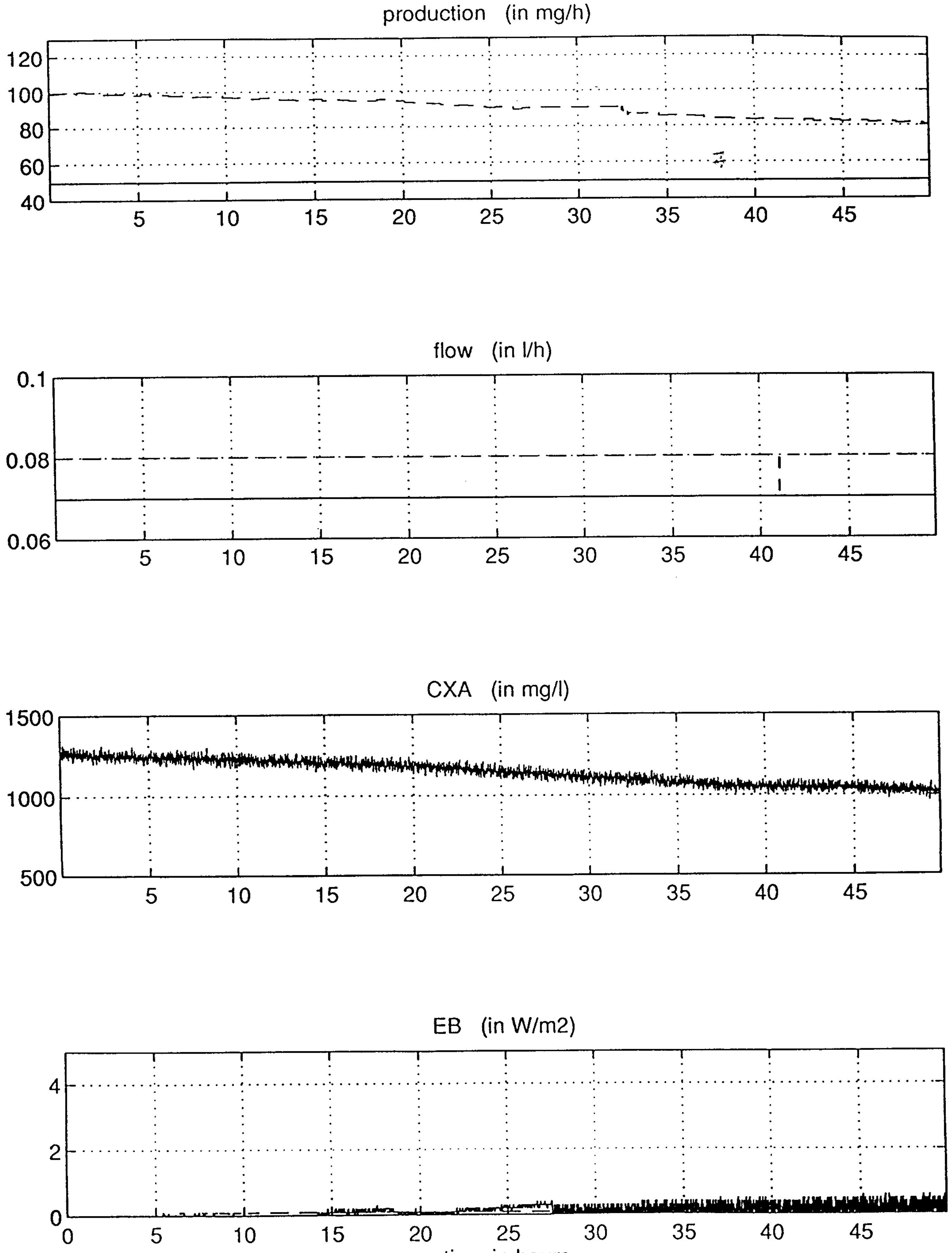


figure 7: experimental results from 03/04 to 05/04

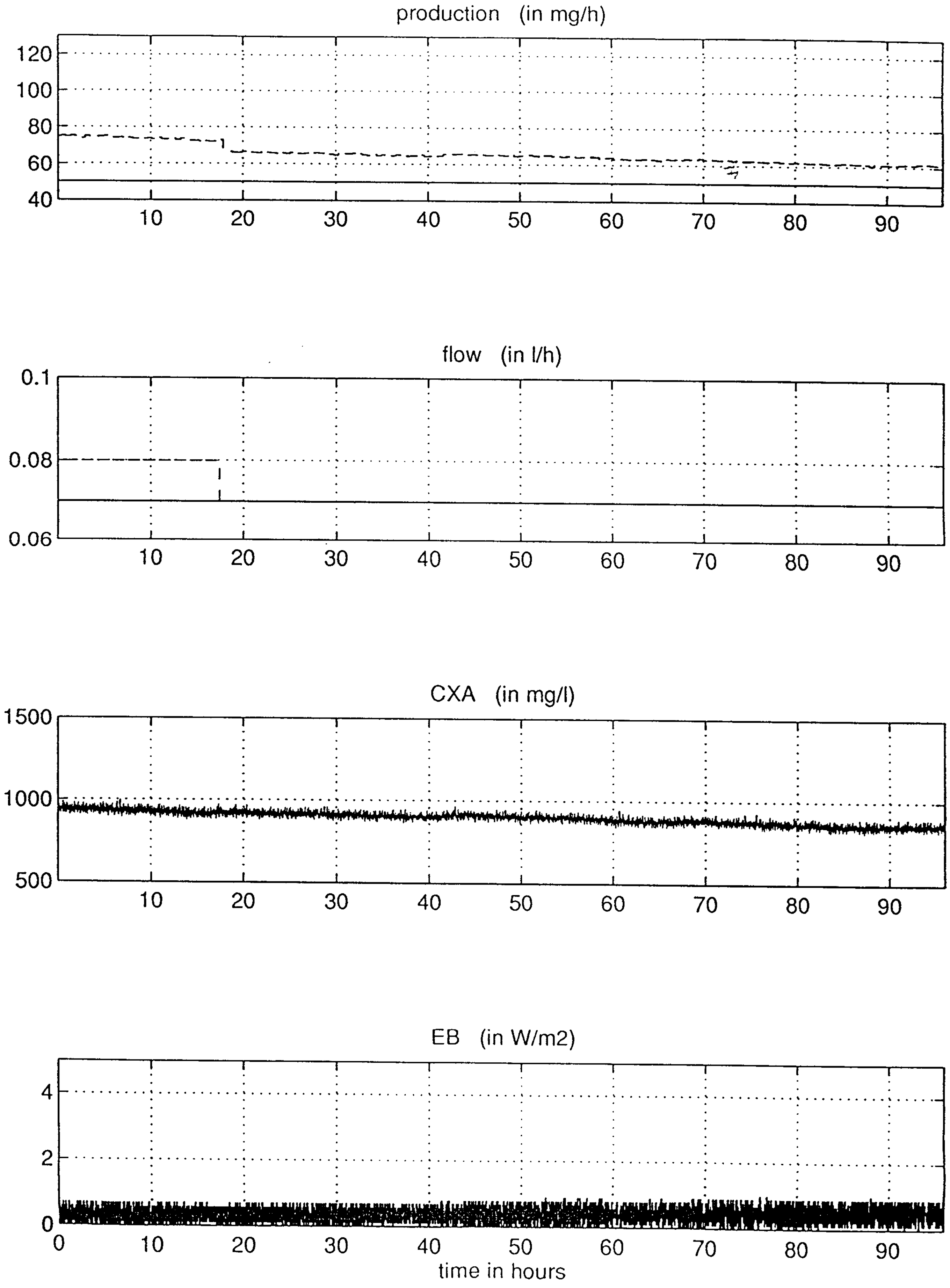


figure 8: experimental results from 06/04 to 09/04

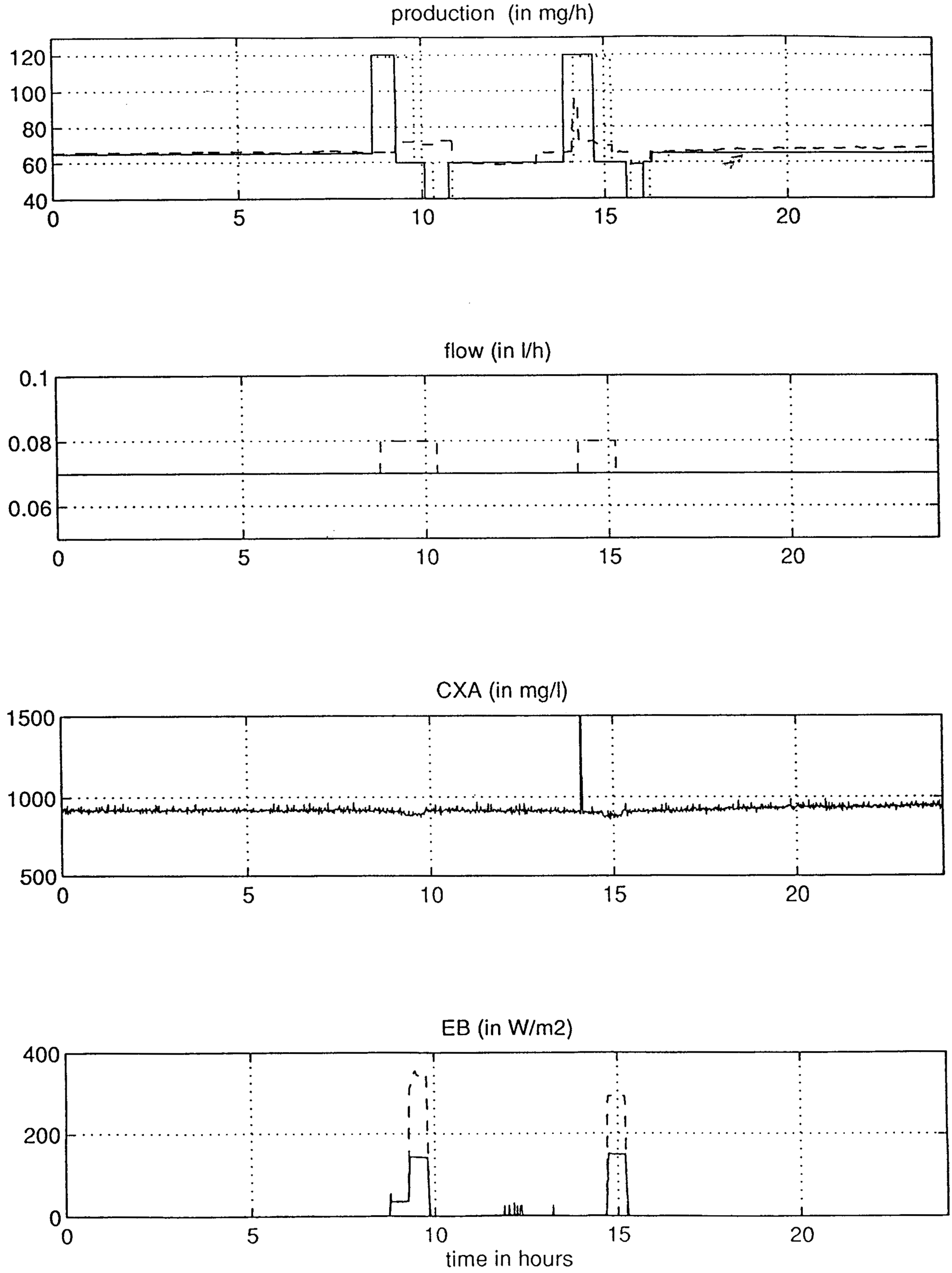
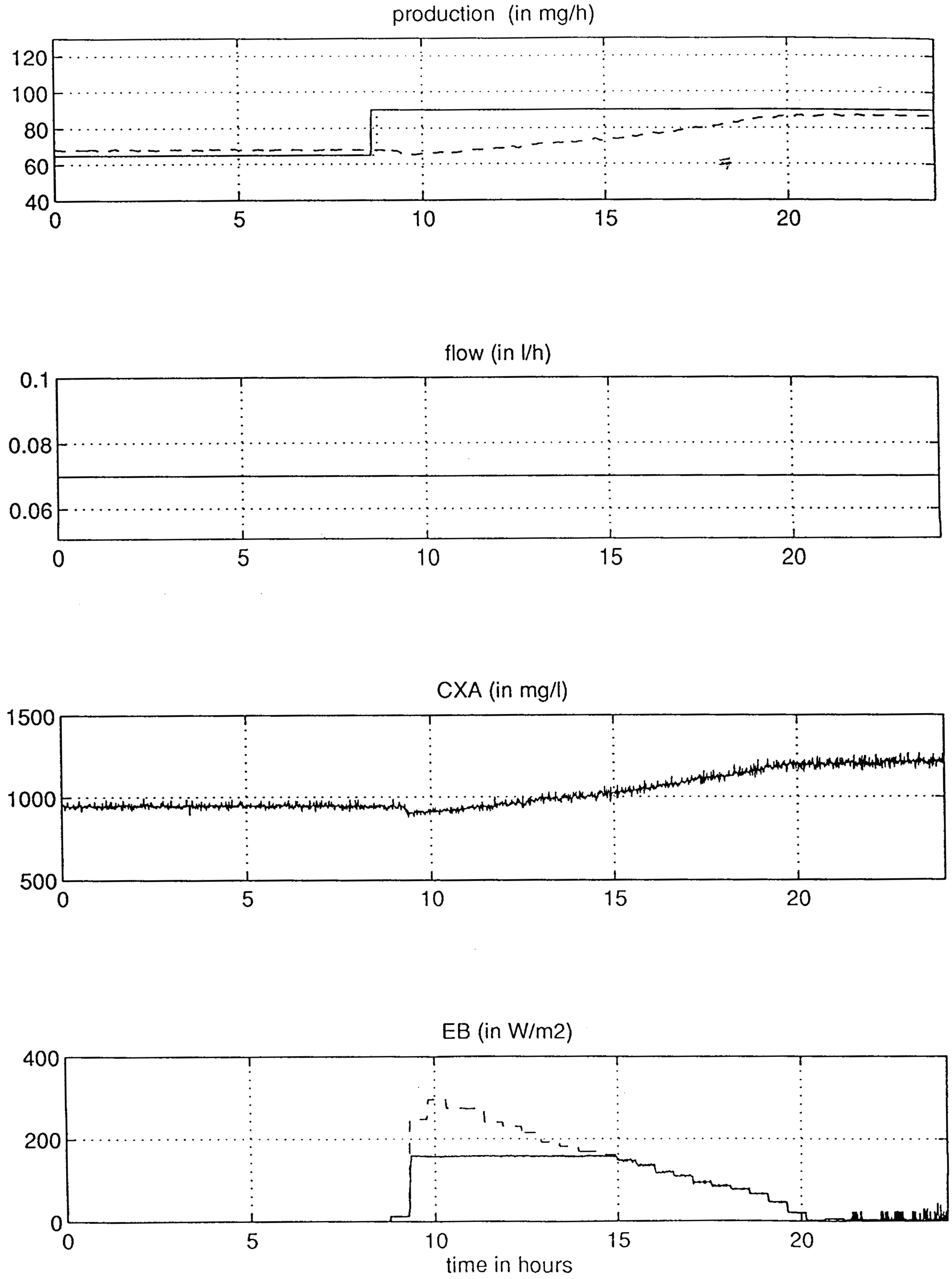


figure 9: experimental results of 12/04


production (in mg/h)

flow (in l/h)

CXA (in mg/l)

EB (in W/m2)

figure 10: experimental results of 13/04

IV - COMPLEMENTARY TESTS

IV.1. Software test

Now, the reactor is installed in Barcelona, at UAB. The software version with predictive control law is called V2.0 (it is the version tested at ESA-ESTEC). This software has been corrected and improved. The new version, named V2.1, was sent at UAB in September.

1 - The main modifications between V2.0 and V2.1 are :

- in Melissa.h

constant values	V2.1	V2.0
VOLUME_TOTAL	7.000 (in l)	7000 (in ml)
VOLUME_LIGHT	3.900 (in l)	3900 (in ml)
FR_MAX	400. (in W/m^2)	8000 (in W/m^2)
CXA_MIN	250. (in mg/l)	500 (in mg/l)
CPT_CXA_UNIT	Suppressed	41.68 (in ml/mn)

- in control.c

- REACT air_lift : suppressed from the global declarations of control.c.

- Flow value display : with 3 digits after the point in V2.1 (2 digits in V2.0) in subroutine "result".

- The subroutine "adersa" modified and renamed "predimod". It is now using the measures C_{N03}, temp, and cxa_moy sended by control-spiru. At that time, the measure C_{N03} is not yet used in the subroutine "model".

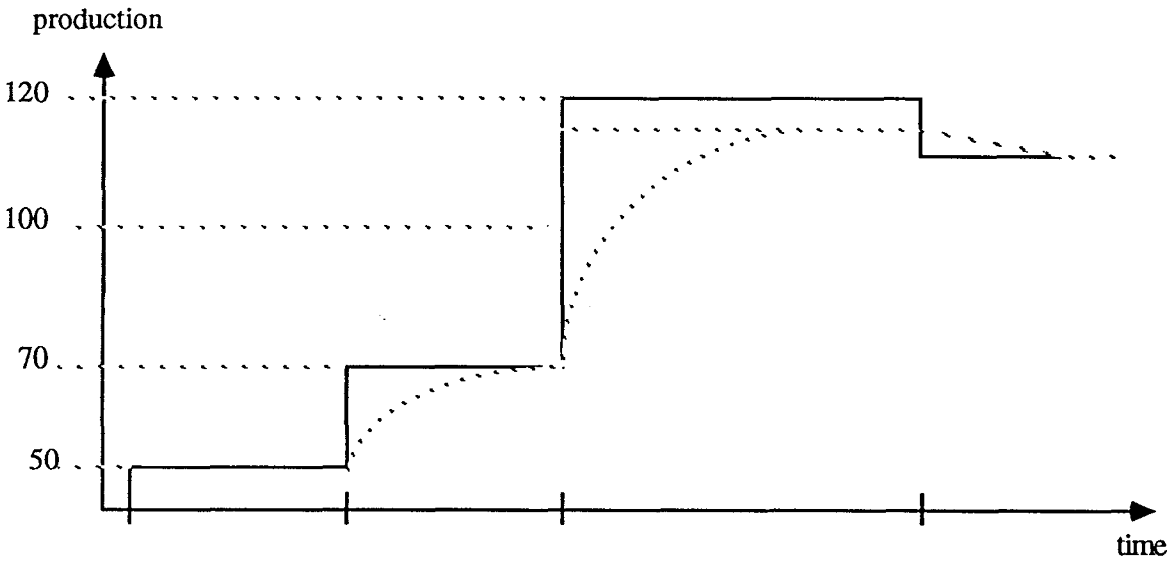
- The subroutine "control-spiru" has been modified to take into account the modifications of the subroutine which calculates the predicted output ("predimod").

- The volume is expressed in l.

- The pump action is now calculated with the calibration coefficient ("cpt-cxa-unit renamed coef-pump) expressed in l/h. This value, in loc_0137, has to be changed because of the unit transformation. The new value is nearly equal to 2.5 (40 ml/mn = 2.4 l/h).

2 - The test to be done, in order to test the software, and PFC strategy is described hereafter :

- batch culture in order to obtain a biomass concentration near 700 mg/l, and then
- nominal setpoint of flow (loc_0151) : 0.07 l/h
- nominal setpoint of production (loc_0150) : 50 mg/h,
  and 70 mg/h,
  and 120 mg/h,
  and 110 mg/h



The variations of setpoint are to be done when the production is stabilized at its new value setpoint (≈ 20 hours ...).

When the nominal production setpoint is equal to 120 mg/h, the concentration setpoint is equal to 1700 mg/l. So, the level 2 will calculate a new flow and a new production setpoint (satisfying the constraints)

qe_real	≈	0.077 l/h
cons_prod_real	≈	115.5 mg/h

- It will be interesting to verify those values in loc_0152 and loc_0153.
- This test doesn't concern the high dilution problem, or the nitrate saturation problem. Other tests to be done in order to study those problems are proposed and defined in IV.2 and IV.3.

3 - Storage

For all the tests, the values that have to be stored in the storage group photosynthèse_ 5 are :

loc_0150	:	Nominal production setpoint (mg/h)
loc_0151	:	Nominal flow setpoint (l/h)
loc_0152	:	Feasible production setpoint (mg/h)
loc_0153	:	Feasible flow setpoint (l/h)
loc_0155	:	measured production (mg/h)
loc_0156	:	model production (mg/h)
loc_0128	:	incident flux setpoint (W/m^2)
SPC_0105	:	setpoint Eb (W/m^2)
PV__0105	:	measured Eb (W/m^2)

in another group, the value of C_{N03}, will have to be stored for the future tests under mineral limitations.

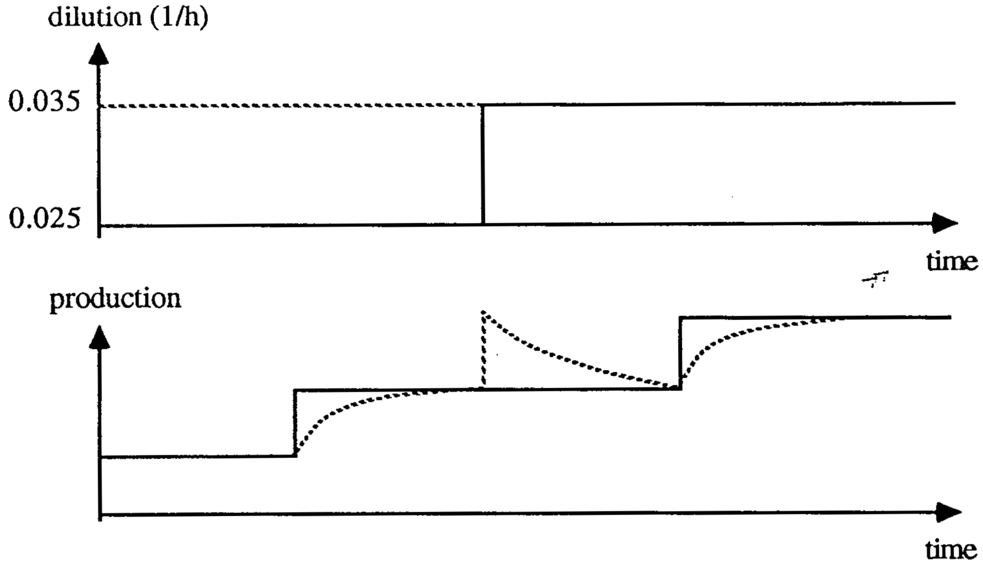
- The results will be sent under ASCII file (1 file per day), on floppy disk (PC-3.5), with a sampling period of 5 minutes.
- In the storage group, the flow values will be stored with 3 digits after the point.

IV.2. High dilution rate

The control law must be tested for high dilution rates.

A first test will be done with *dil* = 0.025 h^{-1}, the corresponding flow is 0.175 l/h. With this dilution rate, a step of production setpoint can be applied from 150 mg/h to 200 mg/h.

Then, a step of dilution rate can be applied, with a new value equal to *dil* = 0.035 h^{-1}. The corresponding flow is 0.245 l/h. With this new value of dilution, a step of production from 200 mg/h to 250 mg/h can be applied when the measured production is stabilized.



This test is realized in closed loop of production, with predictive control law, as the previous test defined in IV.1.

IV.3. Mineral limitation

Other tests will then have to be done to validate the model under Nitrate limitations. In those tests, the Nitrate concentration must be lower than 4 or 5 times the Monod constant. Different dilution rate and different radiant flux values are considered in those tests.

During those tests, the production control loop is open. The setpoint sended to the process are the radiant flux F_r, and the pump flow, proportional to the dilution rate.

These tests have been prepared with the help of Simulink simulator developped at the beginning of the study (from photosim software of LGCB).

With initial concentrations in the reactor :

$C_{XA,\,init} = 0.8$ g/l
$C_{EPS,\,init} = 0.2$ g/l
$C_{N,\,init} = 0.4$ g/l
$C_{S,\,init} = 0.17$ g/l

and concentrations in the input flow :

$C_{XA,\,E} = 0$
$C_{EPS,\,E} = 0$
$C_{N,\,E} = 0.1$ g/l
$C_{E,\,S} = 0.2$ g/l

A first test is proposed with F_r = 50 W/m^2, a second test is proposed with F_r = 200 W/m^2. During those two tests, the dilution rate has the evolution given in figure 11.

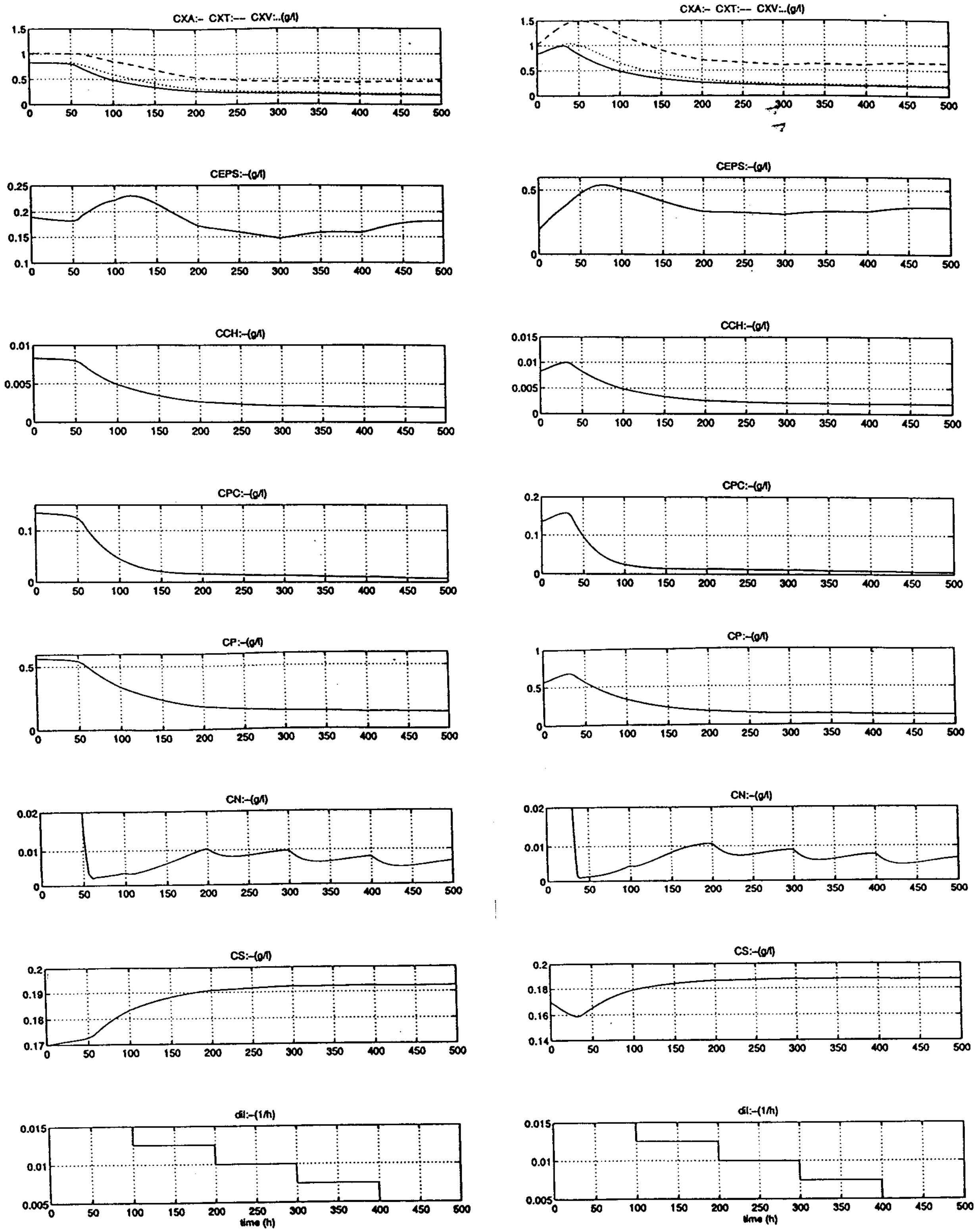


Figure 11 - Simulation results with low nitrate concentration
(a) : F_r = 50 W/m^2 (b) : F_r = 200 W/m^2

Those tests allow to compare the behaviour of the process with the model for different dilution rate and different radiant flux.

During those tests, the biomass and the nitrate concentrations have to be measured on line. But off line measurements have to be done to analyse the quality and the composition of the biomass (C_{PC}, C_P, C_{CH}, C_{XA}, C_G) during those tests. This analysis can also be done for the high dilution tests.

V - CONCLUSION

Now, the first experimental results on Spirulina compartment with non linear predictive control are available and satisfying (the step response time at 95% is nearly equal to 10 or 15 hours, in function of the step value, because of the constraints ; the static precision of production control is smaller than 5%). So the same strategy can certainly be used, without major modifications for the other photosynthetic compartment (rhodobacteria). The LGCB will have to furnish the first principles model adapted to this other bacteria.

In parallel, the global approach for the entire loop will be treated to be able to calibrate the hardware and to define a good global strategy.

REFERENCES

[1] BINOIS C. Control program of Spirulina compartment - ESA-ESTEC Technical Note 18.3 - May 1993

[2] BINOIS C. Automatisation d'un écosystème artificiel utilisé comme système de support vie. Première intéraction modèle/système de contrôle. CNAM Thesis. September 1994.

[3] CORNET J.-F., DUSSAP C.G., GROS J.-B., 1993a. Modelling of physical limitations in photobioreactors. Adaptation of the light energy transfer model to cylindrical geometries. ESA contract PRF 130-820, Technical Note 19.1.

[4] CORNET J.-F., DUSSAP C.G., GROS J.-B., 1993b. Modelling of physical limitations in photobioreactors. Modelling of exopolysaccharide synthesis in cultures of *Spirulina platenis*. ESA contract PRF 130-820, Technical Note 19.2.

[5] CORNET J.-F., DUSSAP C.G., GROS J.-B., 1993c. Modelling of physical limitations in photobioreactors. Applications to simulation and control of the Spirulina Compartment of the MELISSA artifical ecosystem. ESA contract PRF 130-820, Technical Note 19.3.

[6] FULGET N., 1994. MELISSA, first approach of Model Based Predictive Control of Spirulina compartment. ESA contract PRF 132-443, Technical Note 21.1.

[7] FULGET N., 1994. MELISSA, first approach of Model Based Predictive Control of Spirulina compartment. ESA contract PRF 132-443, Technical Note 21.2.

[8] FULGET N., Utilisation d'un programme de simulation CRUE comme élément d'une commande prédictive, rapport ADERSA-CNR.

[9] FULGET N., 1995. MELISSA, Study for the non linear model based predictive control of Spirulina compartment using knowledge model. ESA contract Purchase Order 142356, Technical Note 24.1.

[10] RICHALET J., 1993. Pratique de la Commande Prédictive. HERMES Edition.

[11] RICHALET J. Industrial applications of Predictive Control to rolling mill, fast robot, river dam. ADERSA, internal report.

[12] THAUVOYE O., 1994. MELISSA, Validation d'un modèle mathématique de croissance de bactéries photosynthétiques en photobioréacteur cylindrique. ESA report YCL/2069.0T.

NOTATIONS

E_b	: light measured in the center of the reactor (W/m^2)
F_r	: radiant flux (W/m^2)
C_{XA}	: active biomass concentration (mg/l)
C_{XT}	: total biomass concentration (mg/l)
C_{PC}	: phycocyanin concentration (mg/l)
C_P	: protein concentration (mg/l)
C_{CH}	: chlorophylls concentration (mg/l)
C_G	: glycogen concentration (mg/l)
C_N	: nitrate concentration (mg/l)
C_S	: sulfate concentration (mg/l)
$<r_{XA}>$	: mean growth rate (for active biomass) (mg/l/h)
dil	: dilution rate (1/h)
qe_nom	: nominal pump flow (l/h)
qe_real	: feasible pump flow (l/h)
qe_min	: minimal pump flow (l/h)
qe_max	: maximal pump flow (l/h)
DQ	: maximal variation of flow ($0 < DQ < 1$)
cxa_moy	: average of the 10 last measure of biomass concentration (mg/l)
CXA_MIN	: minimal biomass concentration (mg/l)
CXA_MAX	: maximal concentration (mg/l)
FR_MIN	: minimal radiant flux setpoint (W/m^2)
FR_MAX	: maximal radiant flux setpoint (W/m^2)
prod	: production of active biomass (mg/h)
prod_min	: minimal feasible production (mg/h)
prod_max	: maximal feasible production (mg/h)
cons_prod_real	: feasible setpoint of production (mg/h)
prod_ref	: reference trajectory (futur objective) for production (mg/h)
DT	: sampling period (s)
TR_{95}	: desired time response (at 95%)

Annex A :

C code file used in Matlab simulation

```
#include "comnl.h"
#include "math.h"
#include "proto.h"

/*
   COMNL.C   Algorithme du regulateur non lineaire.
   ESA - MELISSA - SPIRULINE

   Date: 15-DEC-94
*/

/* Declaration des variables statiques */
   static double frmem;


/* ---  COMNL0   -----------------------------------------------------
   Fonction:
      Initialisation du regulateur non lineaire
   Synopsis:
      COMNL0(FR)
   Description:
      Affecte la valeur initiale FR
*/
double comnl0()

{
double   fr ;
fr = frinit;
frmem = fr;
return(fr);
}

/* ---  COMNL   ------------------------------------------------------
   Fonction:
      Equations du regulateur non lineaire
   Synopsis:
      COMNL(CONS_PROD,CXA,QE,FR)
   Description:
      Calcul la commande courante FR a partir de
      la mesure de concentration CXA, de la consigne CONS_PROD, du
      debit QE
*/
double comnl(cons_prod,cxa,qe)

double   cons_prod, cxa, qe ;
{
/* declaration des variables internes */
   double   prod, dil, prod_ref, delfr;
   double   fr , fr1, fr2, prod1, prod2;
   double   qe_max , qe_min , prod_max , prod_min ;

   prod = cxa*qe;

qe_max = qe*(1+dq);
qe_min = qe*(1-dq);
prod_max = qe_max*cxa_max;
prod_min = qe_min*cxa_min;
cons_prod = max(prod_min,min(prod_max,cons_prod));
if (cons_prod/cxa_max > qe )
   {
   qe = min(qe_max,cons_prod/cxa_max);
   }
if (cons_prod/cxa_min < qe )
   {
```

```
   qe = max(qe_min,cons_prod/cxa_min);
   }

   dil = qe/vol;

/* trajectoire de reference */
   prod_ref = cons_prod - pow(lambda,nhc)*(cons_prod - prod);

/* commande precedente */
   fr = frmem;

/* premier scenario */
   frl = fr;
   prodl = model(cxa,frl,dil);

/* deuxieme scenario */
   delfr = dfr*sign(cons_prod - prod);
   fr2 = frl + delfr;
   prod2 = model(cxa,fr2,dil);

/* calcul de fr */
   fr+ = (prod_ref - prodl)/(prod2 - prodl)*delfr;

/* contraintes sur fr */
   fr = max(fr_min,min(fr_max,fr));

/* memorisation */
   frmem=fr;
   return(fr);
}
/* ---  MODEL  ----------------------------------------------------------------
   Fonction:
         integration du modele
   Synopsis:
         MODEL(CXA,FR,DIL,PROD)

*/
double model(cxa,fr,dil)

double cxa, fr, dil ;

{
   double v, dv, vout , prod;
   int k;

   v = cxa;
   for (k=1 ; k <= nhc; k++)
       {
       dv = dercx(v,fr,dil);
       vout =v + dt *dv;
       v=vout;
       }
   prod=vout*dil*vol;
   return (prod);
}


/* --- DERCX ------------------------------------------------------------------

   Fonction:
         calcul de la derivee de cxa
   Synopsis:
         DERCX(cxa,fr,dil,dvt);
```

```
*/
double dercx(cxa,fr,dil)

double cxa, fr, dil;

{
double  dcxdt;
double za, alpha, delta, pij, pijz;
double z, rxa;
za = zpc + zch;

alpha = sqrt(za*Ea/(za*Ea+(1+zg)*Es));
delta = (za*Ea+(1+zg)*Es)*alpha*RT*cxa;

pij = 0.;
for (z=jstep/2; z<=1-jstep/2; z+=jstep)
    {
    pijz = fr/z*2*cosh(delta*z)/(cosh(delta)+alpha*sinh(delta));
    if (pijz>=1)
       {
       pij+ = 2*z*pijz/(Kj+pijz)*jstep;
       }
    }
rxa = muM*pij*zpc*cxa*wiv;

dcxdt = -dil*cxa + rxa ;
return (dcxdt);
}
```

```
#include "math.h"

/* ---   MIN.C   ------------------------------------------------------------
   Fonction:
         Minimum de deux valeurs.
   Synopsis:
         X=MIN(Y,Z)
*/
double min( x1 , x2 )

double x1 , x2;
{
   double  x;
   x = (x1 < x2) ? x1 : x2;
   return( x );
}

/* ---   MAX.C   ------------------------------------------------------------
   Fonction:
         Maximum de deux valeurs.
   Synopsis:
         X=MAX(Y,Z)
*/
double max( x1 , x2 )

double x1 , x2;
{
   double  x;
   x = (x1 > x2) ? x1 : x2;
   return( x );
}

/* ---   SIGN.C   ------------------------------------------------------------
   Fonction:
         signe d'une valeur.
   Synopsis:
         X=SIGN(Y)
*/
double sign( y )

double y;
{
   double  x;
   x = (y < 0) ? -1. : 1.;
   return( x );
}
```

```
/*
   Nom : comnl.h

   Fonction : Coefficients de la commande

   Date : 15-DEC-94
*/

#define dt          0.5       /* control period (in h ) */
#define nhc         5.        /* coincidence point (in dt) */
#define lambda      0.75      /* reference trajectory dynamic */
#define dfr         5.        /* radiant flux increment (in W/m2)  */
#define fr_min      10.       /* min constraint on FR (in W/m2) */
#define fr_max      400.      /* max constraint on FR (in W/m2) */
#define dq          10.       /* flow variation (in %) */
#define cxa_min     0.5       /* min constraint on CXA (in g/l) */
#define cxa_max     1.5       /* max constraint on CXA (in g/l) */
#define vol         7.        /* reactor volume */

#define zpc         .162      /*    */
#define zch         .01       /*    */
#define zg          0.1       /*    */
#define Ea          871.      /*    */
#define Es          167.      /*    */
#define RT          .048      /*    */
#define Kj          20.       /*    */
#define muM         .54       /*    */
#define wiv         .52       /*    */
#define jstep       .01       /*    */

#define frinit      200.      /*    */
```

```
double comnl( );
double comnl0( );
double model( );
double dercx( );
double sign( );
double min( );
double max( );
```

Annex B :

C code file written by C. Binois with PI strategy

(V1.0)

```
/*******************************************************************

        NAME         CONTROL.C

        AUTHOR       BINOIS C

        DESCRIPTION
                CONTROL PROGRAM listing file

        UPDATES
                10-03-93

*******************************************************************/

#include <malloc.h>
#include <math.h>
#include <stdio.h>
#include <stdlib.h>

#include "userdef.h"
#include "melissa.h"

#define SPEED_TIME  60 /* time for measuring growth speed */

int my_interrupt();

/*-----------------------------------
        variables declarations
-----------------------------------*/

VARS   cxa;           /* biomasse concentration          */
VARS   cpt_cxa;       /* biomass regulation counter      */
VARS   Rxa;           /* growth speed of biomasse        */
VARS   Rxa_set_point; /*                                 */

VARS   nitrate;       /* nitrate concentration           */
VARS   cpt_nitrate;   /* nitrate regulation counter      */
VARS   Rnitrate;      /* nitrate consumption rate        */
VARS   cal_nitrate;   /* nitrate calibration switch      */

VARS   light;         /* light intensity in the reactor  */
VARS   temperature;   /* temperature in the reactor      */
VARS   pH;            /* pH of culture                   */
VARS   pressure;      /* gaseous pressure in the reactor */

VARS   Rxa_model;     /* growth speed calculated by model */
VARS   Rxa_error;     /*  error between model and measure */
VARS   error_average;
VARS   error_sigma;
VARS   sum_error;
VARS   cpt_cxa_unit;

VARS   Kp,Ki;
VARS   rxa_filtered;

REACT  air_lift;

char   buffer[100];
```

```
/*------------------------------------
        control programm
------------------------------------*/

void control_spiru()
        {
        acq_vars();
        calc_rx();
        air_lift.Cxa=cxa.value;
        air_lift.Cno3=nitrate.value;
        air_lift.temp=temperature.value;
        air_lift.press=pressure.value;
        air_lift.Eb=light.value;
        model(&air_lift);
/*
        filter();       supprimé pour pouvoir travailler en vitesse
*/                                                        uniquement -
        Rxa_model.sp=rxa_filtered.sp
                        +Kp.value*Rxa_error.sp+Ki.value*sum_error.sp;
        solver(&air_lift,Rxa_model.sp);
        light.sp=air_lift.Eb;
        Rxa_model.sp=air_lift.rxa;

        send_vars();
        result();
        }


/*------------------------------------
        mathematical model
------------------------------------*/

model(REACT *react)
        {
        double  zpc=.135;
        double  zp=.57;
        double  zch=0.0085;
        double  zg=0;
        double  za;
        double  Ea=871;
        double  Es=167;
        double  alpha,delta,delta3;
        double  Fr,Fr1,Fr2,Fr3;
        double  R,R1,R2,R3,Rb;
        double  z;
        double  jstep=0.01;
        double  pij,pijz;
        double  Kj=20;
        double  KN=5.3;
        double  muM=0.54;
        double  yn=0.42;

        double  coeft,coefN,Rmean;
        R=0.048;
        R1=0.0302;
        R2=0.02585;
        R3=0.0115;
        Rb=0.0095;

/*  general parameters ----------------------------------*/

        za=zpc+zch;
        alpha=sqrt(za*Ea/(za*Ea+(1+zg)*Es));
        delta=(za*Ea+(1+zg)*Es)*react->Cxa/1000*alpha*R;
        delta3=delta*R2/R;

/*  incident flux determination -------------------------*/

        Fr3=react->Eb*Rb/PI/R3;
        z=R3/R2;
        Fr2=Fr3*z/(2*alpha)*(cosh(delta3)+alpha*sinh(delta3))/sinh(delta3*z);
        Fr1=Fr2*R2/R1;
        z=R1/R;
        Fr=Fr1*z/(2*alpha)*(cosh(delta)+alpha*sinh(delta))/sinh(delta*z);
        react->Fr=Fr;

/*  determination of the mean growth rate ---------------*/

        pij=0;
        for(z=jstep/2;z<=1-jstep/2;z+=jstep)
                {
                if((z<R2/R)||(z>R1/R))
                        {
                        pijz=Fr/z*2*cosh(delta*z)/(cosh(delta)+alpha*sinh(delta));
                        if(pijz>=1)
                                {
                                pij+=2*z*pijz/(Kj+pijz)*jstep;
                                }
                        }
                }
        Rmean=muM*pij*zpc*react->Cxa*VOLUME_LIGHT/VOLUME_TOTAL;

/* temperature and nitrates correction */

        coeft=0.8*exp(-pow((react->temp-35)/10,2))+0.2;
        if(react->Cno3<20)
                {
                react->Cno3=100;
                }
        coefN=react->Cno3/(KN+react->Cno3);
coefN=1;
        react->rxa=Rmean*coefN*coeft;
        react->rn=yn*react->rxa;
        }


/*-----------------------------------
        solver using model
-----------------------------------*/

solver(REACT *react,double Rx_seek)
        {
        char    buffer[100];
        REACT   *react1,*react2,*react_seek,*preact;
```

```
display_status("Model working ...");
model(react);
react->rxa=(react->rxa<0.2)? 0.2 : react->rxa;
Rx_seek=(Rx_seek<=0.2)? 0.2 : Rx_seek;
if(fabs(react->rxa-Rx_seek)<ERROR_SPEED)
        {
        display_status(" ");
        return(0);
        }
react1=malloc(sizeof(REACT));
react2=malloc(sizeof(REACT));
react_seek=malloc(sizeof(REACT));

copy_react(react,react1);
copy_react(react,react2);
copy_react(react,react_seek);
react1->Eb=0;
react1->rxa=0;
while(Rx_seek>react2->rxa)
        {
        react2->Eb+=50;
        if(react2->Eb>100)
                {
                react2->Eb=100;
                model(react2);
                Rx_seek=react2->rxa-0.2;
                break;
                }
        model(react2);
        }
while(fabs(react_seek->rxa-Rx_seek)>ERROR_SPEED)
        {
        react_seek->Eb=(react1->Eb+react2->Eb)/2;
        model(react_seek);
        if(Rx_seek>react_seek->rxa)
                {
                preact=react1;
                react1=react_seek;
                react_seek=preact;
                }
        else
                {
                preact=react2;
                react2=react_seek;
                react_seek=preact;
                }
        if(fabs(react1->Eb-react2->Eb)<0.001)
                {
                break;
                }
        }
copy_react(react_seek,react);
free(react_seek);
free(react1);
free(react2);
display_status(" ");
}
```

```
/*------------------------------------
        copy structure reacta to reactb
-------------------------------------*/

copy_react(REACT *reacta,REACT *reactb)

        {
        reactb->Cxa=reacta->Cxa;
        reactb->Cno3=reacta->Cno3;
        reactb->temp=reacta->temp;
        reactb->press=reacta->press;

        reactb->Eb=reacta->Eb;
        reactb->Fr=reacta->Fr;
        reactb->rxa=reacta->rxa;
        reactb->rn=reacta->rn;
        reactb->ro2=reacta->ro2;
        }


/*------------------------------------
        growth speed calculation
-------------------------------------*/

calc_rx()
        {
        double  diff_cpt(VARS *,int);
        double  average_var(VARS *,int);
        double  average2_var(VARS *,int);
        double  slope_var(VARS *,int);
        double  slope_cpt(VARS *,int);
        double  dcxa_dt;


        dcxa_dt=slope_var(&cxa,SPEED_TIME);
        Rxa.sp=(slope_cpt(&cpt_cxa,SPEED_TIME)*cpt_cxa_unit.value/VOLUME_TOTAL
        *cxa.value)*60/SPEED_TIME;

        Rxa_error.sp=Rxa_set_point.value-Rxa.sp;
        sum_error.sp+=Rxa_error.sp;
        error_average.sp=average_var(&Rxa_error,240);
        error_sigma.sp=sqrt(average2_var(&Rxa_error,240)-pow(error_average.sp,2));
        }


/*------------------------------------
        filter for the reference compensation
-------------------------------------*/

filter()

        {
        static double T[3];
        double p[3],q[3],k;

        k=0.0316227766;
        p[1]=1.96255627;
        p[2]=-0.96327226;
        q[0]=1;
        q[1]=-0.014618827;
        q[2]=0.0367277331;

        T[0]=k*Rxa_set_point.sp+p[1]*T[1]+p[2]*T[2];
        T[2]=T[1];
        T[1]=T[0];
        rxa_filtered.sp=k*Rxa_set_point.sp*q[0]+q[1]*T[1]+q[2]*T[2];
        }


/*-----------------------------------
        variables initialisation
-----------------------------------*/

init_vars()
        {
        REACT  init_react;
        double  delta;
        int     jj;

        display_status("Initialisation of variables ...");

/*  TAG and COMMAND name initialisation     */

        sprintf(cxa.name,"LOOP0107");
        sprintf(cpt_cxa.name,"LOC-0111");
        sprintf(Rxa.name,"LOC-0130");
        sprintf(Rxa_set_point.name,"LOC-0132");

        sprintf(nitrate.name,"LOOP0103");
        sprintf(cpt_nitrate.name,"LOC-0109");
        sprintf(Rnitrate.name,"LOC-0133");
        sprintf(cal_nitrate.name,"DI--0125");

        sprintf(light.name,"LOOP0105");
        sprintf(temperature.name,"LOOP0106");
        sprintf(pH.name,"LOOP0104");
        sprintf(pressure.name,"LOOP0102");

        sprintf(Rxa_model.name,"LOC-0131");
        sprintf(Rxa_error.name,"LOC-0133");
        sprintf(error_average.name,"LOC-0134");
        sprintf(error_sigma.name,"LOC-0135");
        sprintf(sum_error.name,"LOC-0136");
        sprintf(cpt_cxa_unit.name,"LOC-0137");
        sprintf(Kp.name,"LOC-0138");
        sprintf(Ki.name,"LOC-0139");
        sprintf(rxa_filtered.name,"LOC-0140");

/*  Variables and Counters initialisation   */
        acq_vars();
        init_react.Cxa=cxa.value;
        init_react.Cno3=nitrate.value;
        init_react.temp=temperature.value;
        init_react.press=pressure.value;
        init_react.Eb=light.value;
        model(&init_react);
        Rxa_set_point.sp=init_react.rxa;
        write_var(&Rxa_set_point);

        delta=-init_react.rxa/init_react.Cxa*VOLUME_TOTAL/cpt_cxa_unit.value
        *TSAMP/3600;
        fill_struct_cpt(&cpt_cxa,delta);
        fill_struct_cpt(&cpt_nitrate,0.42*delta);
        fill_struct_var(&cxa);

        Rxa_error.value=0;
        fill_struct_var(&Rxa_error);
        sum_error.value=0;
        fill_struct_var(&sum_error);
        fill_struct_var(&Kp);
        fill_struct_var(&Ki);

/* filter initialisation */

        for(jj=0;jj<500;jj++)
                {
                filter();
                }
        write_var(&rxa_filtered);

        wait_time(1);
        display_status(" ");
        }

/*-----------------------------------
        variables acquisition
-----------------------------------*/

acq_vars()
        {
        display_status("Acquisition of variables ...");

        read_var(&cxa);
        if(fabs(cxa.value-cxa.sp)>20)
                {
                cxa.value=cxa.sp;
                cxa.val[cxa.i]=cxa.sp;
                }
        read_var(&cpt_cxa);
        read_var(&Rxa);
        read_var(&Rxa_set_point);
        read_var(&cpt_cxa_unit);

/*  nitrate analyser calibration  */

        read_var(&cal_nitrate);
        if(!cal_nitrate.value)
                {
                read_var(&nitrate);
                read_var(&cpt_nitrate);
                }

        read_var(&Rnitrate);
        read_var(&light);
        read_var(&temperature);
        read_var(&pH);
        read_var(&pressure);

        read_var(&Rxa_model);
        read_var(&Rxa_error);
        read_var(&error_average);
        read_var(&error_sigma);
        read_var(&sum_error);
        read_var(&Ki);
        read_var(&Kp);


        read_var(&rxa_filtered);

        display_status(" ");
        }

/*------------------------------------
        commands updating
------------------------------------*/

send_vars()
        {
        display_status("Updating variables ...");

        write_var(&Rxa);
        write_var(&Rxa_set_point);

        write_var(&Rnitrate);
        write_var(&light);

        write_var(&Rxa_model);
        write_var(&Rxa_error);
        write_var(&error_average);
        write_var(&error_sigma);
        write_var(&sum_error);
        write_var(&rxa_filtered);
        display_status(" ");
        }

/*------------------------------------
        prepare result of control for display
------------------------------------*/

result()
        {
        void    display_result(char *,short,short);
        char buffer[150];

        sprintf(buffer,"Concentrations  Biomass");
        display_result(buffer,1,1);
        sprintf(buffer,"mg/l");
        display_result(buffer,32,1);
        display_result(buffer,32,2);
        sprintf(buffer,"Nitrate");
        display_result(buffer,17,2);
        sprintf(buffer,"%.1f",cxa.value);
        display_result(buffer,26,1);
        sprintf(buffer,"%.1f",nitrate.value);
        display_result(buffer,26,2);

        sprintf(buffer,"Light");
        display_result(buffer,1,4);
        sprintf(buffer,"Eb      W/m2");
        display_result(buffer,22,4);
        sprintf(buffer,"Fr      W/m2");
        display_result(buffer,22,5);
        sprintf(buffer,"%.1f",light.value);
        display_result(buffer,26,4);
        sprintf(buffer,"%.1f",air_lift.Fr);
        display_result(buffer,26,5);

        sprintf(buffer,"Kinetics     measured       mg/lh");
        display_result(buffer,1,7);
        sprintf(buffer,"%.2f",Rxa.sp);
        display_result(buffer,26,7);

        sprintf(buffer,"set-point      mg/lh");
        display_result(buffer,15,8);
        sprintf(buffer,"%.2f",Rxa_set_point.value);
        display_result(buffer,26,8);

        sprintf(buffer,"model      mg/lh");
        display_result(buffer,19,9);
        sprintf(buffer,"%.2f",Rxa_model.sp);
        display_result(buffer,26,9);

        sprintf(buffer,"error      mg/lh");
        display_result(buffer,19,10);
        sprintf(buffer,"%.2f",Rxa_error.sp);
        display_result(buffer,26,10);

        sprintf(buffer,"average error      mg/lh");
        display_result(buffer,11,11);
        sprintf(buffer,"%.2f",error_average.sp);
        display_result(buffer,26,11);

        sprintf(buffer,"sigma error      mg/lh");
        display_result(buffer,13,12);
        sprintf(buffer,"%.2f",error_sigma.sp);
        display_result(buffer,26,12);

        }


/*------------------------------------
        calculate the delta count during time t in minutes
------------------------------------*/

double diff_cpt(VARS *diff_var, int diff_time)
        {
        int j;
        int i_samp,i_prev,nb_samp;
        double total_count;

        total_count=0;
        nb_samp=ceil(diff_time*60/TSAMP);
        for(j=0;j<nb_samp;j++)
                {
                i_samp=(diff_var->i-j)&NB_SAMP;
                i_prev=(i_samp-1)&NB_SAMP;
                total_count+= ( diff_var->val[i_samp]>=diff_var->val[i_prev]) ?
                diff_var->val[i_samp]-diff_var->val[i_prev] : diff_var->val[i_samp];
                }
        return(total_count);
        }


/*-----------------------------------
        calculate the variable variation during time t in minutes
-----------------------------------*/

double diff_var(VARS *diff_var, int diff_time)
        {
        double dvar_dt;
        int nb_samp;

        nb_samp=ceil(diff_time*60/TSAMP);
        dvar_dt=diff_var->val[diff_var->i]-diff_var->val[(diff_var->i
        -nb_samp)&NB_SAMP];
        return(dvar_dt);
        }

/*-----------------------------------
        calculate the average during time t in minutes
-----------------------------------*/

double average_var(VARS *diff_var, int diff_time)
        {
        int j;
        int i_samp,nb_samp;
        double average;

        average=0;
        nb_samp=ceil(diff_time*60/TSAMP);
        for(j=0;j<nb_samp;j++)
                {
                i_samp=(diff_var->i-j)&NB_SAMP;
                average+=diff_var->val[i_samp];
                }
        average/=nb_samp;
        return(average);
        }

/*-----------------------------------
        calculate the average^2 during time t in minutes
-----------------------------------*/

double average2_var(VARS *diff_var, int diff_time)
        {
        int j;
        int i_samp,nb_samp;
        double average;

        average=0;
        nb_samp=ceil(diff_time*60/TSAMP);
        for(j=0;j<nb_samp;j++)
                {
                i_samp=(diff_var->i-j)&NB_SAMP;
                average+=pow(diff_var->val[i_samp],2);
                }
        average/=nb_samp;
        return(average);
        }

/*------------------------------------
        fill val[i] with the current value
------------------------------------*/

fill_struct_var(VARS *fill_struct)

        {
        int jj;

        for(jj=0;jj<=NB_SAMP;jj++)
                {
                fill_struct->val[jj]=fill_struct->value;
                }
        }


/*------------------------------------
        fill val[i] with the current value and delta between each value
------------------------------------*/

fill_struct_cpt(VARS *fill_struct,double _delta)

        {
        int jj,kk,ll;

        for(jj=0;jj<NB_SAMP;jj++)
                {
                kk=(fill_struct->i-jj)&NB_SAMP;
                ll=(kk-1)&NB_SAMP;
                fill_struct->val[ll]=fill_struct->val[kk]+_delta;
                }
        }


/*------------------------------------
        calculate the slope of variable by the least mean square method
------------------------------------*/

double slope_var(VARS *slope_var,int diff_time)
        {
        int ii,jj,kk;
        int nb_samp;
        double slope;
        double sumxi, sumyi, sumxiyi, sumxi2;
        sumxi=0;
        sumyi=0;
        sumxiyi=0;
        sumxi2=0;

        nb_samp=ceil(diff_time*60/TSAMP);
        for(ii=0;ii<nb_samp;ii++)
                {
                jj=slope_var->i-ii;
                kk=(slope_var->i-ii)&NB_SAMP;
                sumxi+=jj;
                sumyi+=slope_var->val[kk];
                sumxiyi+=jj*slope_var->val[kk];
                sumxi2+=pow((double)jj,2);
                }
        slope=nb_samp*(nb_samp*sumxiyi-sumxi*sumyi)/(nb_samp*sumxi2-sumxi*sumxi);
        return(slope);
        }


/*-----------------------------------
        calculate the slope of counter by the least mean square method
-----------------------------------*/

double slope_cpt(VARS *slope_cpt,int diff_time)
        {
        int ii,jj,kk,ll;
        int nb_samp;
        double slope;
        double sumxi, sumyi, sumxiyi, sumxi2;
        double raz_cpt;

        raz_cpt=0;
        sumxi=0;
        sumyi=0;
        sumxiyi=0;
        sumxi2=0;

        nb_samp=ceil(diff_time*60/TSAMP);
        for(ii=0;ii<nb_samp;ii++)
                {
                jj=slope_cpt->i-ii;
                kk=(slope_cpt->i-ii)&NB_SAMP;
                ll=(kk-1)&NB_SAMP;
                sumxi+=jj;
                sumyi+=(slope_cpt->val[kk]-raz_cpt);
                sumxiyi+=jj*(slope_cpt->val[kk]-raz_cpt);
                sumxi2+=pow((double)jj,2);
                if(slope_cpt->val[ll]>slope_cpt->val[kk])
                        {
                        raz_cpt=slope_cpt->val[ll];
                        }
                }
        slope=nb_samp*(nb_samp*sumxiyi-sumxi*sumyi)/(nb_samp*sumxi2-sumxi*sumxi);
        return(slope);
        }
```

Annex C :

C code file with non linear predictive control

(V2.0)

```
/*********************************************************************

        NAME            ALARMS.C

        AUTHOR          BINOIS C

        DESCRIPTION
                ALARM management listing file

        UPDATES
                05-05-93

*********************************************************************/

#include <math.h>
#include <stdio.h>
#include <stdlib.h>

#include "userdef.h"
#include "melissa.h"

/*---------------------------------------
        variables declarations
---------------------------------------*/


char buffer[100];

/*---------------------------------------
        alarm programm
---------------------------------------*/

void alarm_spiru()
        {
        acq_alarm();
        send_alarm();
        }

/*---------------------------------------
        alarms initialisation
---------------------------------------*/

init_alarm()
        {
        display_status("Initialisation of ALARMS ...");
        wait_time(2);
        display_status(" ");
        }

/*---------------------------------------
        alarms acquisition
---------------------------------------*/

acq_alarm()
        {
        display_status("Acquisition of ALARMS ...");

        wait_time(2);
        display_status(" ");
        }

/*---------------------------------------
        alarms updating
---------------------------------------*/

send_alarm()
        {
        display_status("Updating ALARMS ...");

        wait_time(2);
        display_status(" ");
        }
```

```
/*********************************************************************

        NAME            CONTROL.C

        AUTHOR          BINOIS C

        DESCRIPTION
                CONTROL PROGRAM listing file

        UPDATES
                10-03-93

*********************************************************************/

#include <malloc.h>
#include <math.h>
#include <stdio.h>
#include <stdlib.h>

#include "userdef.h"
#include "melissa.h"

#define SPEED_TIME  60  /* time for measuring growth speed */

int my_interrupt();

/*-------------------------------------
        variables declarations
-------------------------------------*/

VARS    cxa;            /* biomasse concentration           */
VARS    nitrate;        /* nitrate concentration            */
VARS    cal_nitrate;    /* nitrate calibration switch       */

VARS    Eb;             /* light intensity in the reactor   */
VARS    Fr;             /* incident flux                    */
VARS    temperature;    /* temperature in the reactor       */
VARS    pH;             /* pH of culture                    */
VARS    act_pompe;      /* action pompe de dilution         */
VARS    cpt_cxa_unit;    /* calibration of pump ml/cpt       */

/********************   variables ADERSA   *****************/

VARS    cons_prod_nom;  /* consigne de production           */
VARS    cons_prod_real; /* consigne realisee                */
VARS    qe_nom;         /* consigne de debit                */
VARS    qe_real;        /* consigne realisee                */
VARS    production;     /* production realisee              */
VARS    prod_mod;       /* production modele                */

REACT   air_lift;
int     next_pfc;       /* next execution of PFC            */
char    buffer[100];

/*-------------------------------------
        mathematical model
-------------------------------------*/

model(REACT *react, int mode)
   {

        double  zpc=.135;
        double  zp=.57;
        double  zch=0.0085;
        double  zg=0;
        double  za;
        double  Ea=871;
        double  Es=167;
        double  alpha,delta,delta3;
        double  Fr,Fr1,Fr2,Fr3;
        double  R,R1,R2,R3,Rb;
        double  z;
        double  jstep=0.01;
        double  pij,pijz;
        double  Kj=20;
        double  KN=5.3;
        double  muM=0.54;
        double  yn=0.42;

        double  coeft,coefN,Rmean;
        R=0.048;
        R1=0.0302;
        R2=0.02585;
        R3=0.0115;
        Rb=0.0095;

/*  general parameters ---------------------------------*/
```

```
        za=zpc+zch;
        alpha=sqrt(za*Ea/(za*Ea+(1+zg)*Es));
        delta=(za*Ea+(1+zg)*Es)*react->Cxa/1000*alpha*R;
        delta3=delta*R2/R;

        switch(mode) {

                case USE_EB:
                {
                /*  incident flux determination --------------*/
                Fr3=react->Eb*Rb/PI/R3;
                z=R3/R2;
                Fr2=Fr3*z/(2*alpha)*(cosh(delta3)+alpha*sinh(delta3))/sinh(delta3*z);
                Fr1=Fr2*R2/R1;
                z=R1/R;
                Fr=Fr1*z/(2*alpha)*(cosh(delta)+alpha*sinh(delta))/sinh(delta*z);
                react->Fr=Fr;
                break;
                }

                case USE_FR:
                {
                /*  Eb determination -------------------------*/
                z=R1/R;
                Fr=react->Fr;
                Fr1=Fr/(z/(2*alpha)*(cosh(delta)+alpha*sinh(delta))/sinh(delta*z));
                Fr2=Fr1/(R2/R1);
                z=R3/R2;
                Fr3=Fr2/(z/(2*alpha)*(cosh(delta3)+alpha*sinh(delta3))/sinh(delta3*z));
                react->Eb=Fr3/(Rb/PI/R3);
                break;
                }

                default:
                {
                display_error("Incorrect model call ...\n");
                display_error("*** Program terminated ***\n");
                exit(0);
                }
        }

/*  determination of the mean growth rate ---------------*/

        pij=0;
        for(z=jstep/2;z<=1-jstep/2;z+=jstep)
                {
                if((z<R2/R)||(z>R1/R))
                        {
                        pijz=Fr/z*2*cosh(delta*z)/(cosh(delta)+alpha*sinh(delta));
                        if(pijz>=1)
                                {
                                pij+=2*z*pijz/(Kj+pijz)*jstep;
                                }
                        }
                }
        Rmean=muM*pij*zpc*react->Cxa*VOLUME_LIGHT/VOLUME_TOTAL;

/* temperature and nitrates correction */

        coeft=0.8*exp(-pow((react->temp-35)/10,2))+0.2;
        coefN=react->Cno3/(KN+react->Cno3);


/******  nitrate saturation  ********/
coefN=1;
/************************************/
        react->rxa=Rmean*coefN*coeft;
        react->rn=yn*react->rxa;


        }

/*-------------------------------------
        copy structure reacta to reactb
-------------------------------------*/

copy_react(REACT *reacta,REACT *reactb)

        {
        reactb->Cxa=reacta->Cxa;
        reactb->Cno3=reacta->Cno3;
        reactb->temp=reacta->temp;
        reactb->press=reacta->press;

        reactb->Eb=reacta->Eb;
        reactb->Fr=reacta->Fr;
        reactb->rxa=reacta->rxa;
        reactb->rn=reacta->rn;
        reactb->ro2=reacta->ro2;
        }


/*------------------------------------
        variables initialisation
-------------------------------------*/

init_vars()
        {
        REACT   init_react;
        double  delta;
        int     jj;

        display_status("Initialisation of variables ...");

/*  TAG and COMMAND name initialisation     */

        sprintf(cxa.name,"LOOP0107");
        sprintf(nitrate.name,"LOOP0103");
        sprintf(cal_nitrate.name,"DI--0125");

        sprintf(Eb.name,"LOOP0105");
        sprintf(Fr.name,"LOC-0128");
        sprintf(temperature.name,"LOOP0106");
        sprintf(pH.name,"LOOP0104");
        sprintf(act_pompe.name,"LOC-0154");
        sprintf(cpt_cxa_unit.name,"LOC-0137");

        sprintf(cons_prod_nom.name,"LOC-0150");
        sprintf(cons_prod_real.name,"LOC-0152");
        sprintf(qe_nom.name,"LOC-0151");
        sprintf(qe_real.name,"LOC-0153");
        sprintf(production.name,"LOC-0155");
        sprintf(prod_mod.name,"LOC-0156");

/*  Variables initialisation   */

        acq_vars();

        init_react.Cxa=cxa.value;
        init_react.Cno3=nitrate.value;
        init_react.temp=temperature.value;
        init_react.Eb=Eb.value;
        model(&init_react,USE_EB);
        Fr.sp=init_react.Fr;
        write_var(&Fr);
        fill_struct_var(&cxa);

        cons_prod_real.sp=cons_prod_nom.value;
        write_var(&cons_prod_real);
        qe_real.sp=qe_nom.value;
        write_var(&qe_real);

/* initialisation timer PFC */
        next_pfc=DT;

        wait_time(1);

        display_status(" ");
        }

/*------------------------------------
        variables acquisition
-------------------------------------*/

acq_vars()
        {
        display_status("Acquisition of variables ...");

        read_var(&cxa);

/*  nitrate analyser calibration  */

        read_var(&cal_nitrate);
        if(!cal_nitrate.value)
                {
                read_var(&nitrate);
                }

        read_var(&Eb);
        read_var(&Fr);
        read_var(&temperature);
        read_var(&pH);
        read_var(&act_pompe);
        read_var(&cpt_cxa_unit);
        read_var(&cons_prod_nom);
        read_var(&cons_prod_real);
        read_var(&qe_nom);
        read_var(&qe_real);
        read_var(&production);
        read_var(&prod_mod);

        display_status(" ");
        }

/*-------------------------------------
        commands updating
-------------------------------------*/

send_vars()
        {
        display_status("Updating variables ...");

        write_var(&Eb);
        write_var(&Fr);
        write_var(&act_pompe);
        write_var(&cons_prod_real);
        write_var(&qe_real);
        write_var(&production);
        write_var(&prod_mod);

        display_status(" ");
        }

/*-------------------------------------
        prepare result of control for display
-------------------------------------*/

result()
        {
        void display_result(char *,short,short);
        char buffer[150];

        sprintf(buffer,"Concentrations  Biomass");
        display_result(buffer,1,1);
        sprintf(buffer,"mg/l");
        display_result(buffer,32,1);
        display_result(buffer,32,2);
        sprintf(buffer,"Nitrate");
        display_result(buffer,17,2);
        sprintf(buffer,"%.1f",cxa.value);
        display_result(buffer,26,1);
        sprintf(buffer,"%.1f",nitrate.value);
        display_result(buffer,26,2);

        sprintf(buffer,"Light");
        display_result(buffer,1,4);
        sprintf(buffer,"Eb        W/m2");
        display_result(buffer,22,4);
        sprintf(buffer,"Fr        W/m2");
        display_result(buffer,22,5);
        sprintf(buffer,"%.1f",Eb.sp);
        display_result(buffer,26,4);
        sprintf(buffer,"%.1f",Fr.sp);
        display_result(buffer,26,5);

        sprintf(buffer,"Production     measured         mg/h");
        display_result(buffer,1,7);
        sprintf(buffer,"%.2f",production.sp);
        display_result(buffer,26,7);

        sprintf(buffer,"set-point        mg/h");
        display_result(buffer,15,8);
        sprintf(buffer,"%.2f",cons_prod_nom.value);
        display_result(buffer,26,8);

        sprintf(buffer,"realised        mg/h");
        display_result(buffer,16,9);
        sprintf(buffer,"%.2f",cons_prod_real.sp);
        display_result(buffer,26,9);

        sprintf(buffer,"model         mg/h");
        display_result(buffer,15,10);
        sprintf(buffer,"%.2f",prod_mod.sp);
        display_result(buffer,26,10);

        sprintf(buffer,"Flow           realised         l/h");
        display_result(buffer,1,11);
        sprintf(buffer,"%.2f",qe_real.sp);
        display_result(buffer,26,11);

        sprintf(buffer,"set point        l/h");
        display_result(buffer,15,12);
        sprintf(buffer,"%.2f",qe_real.value);
        display_result(buffer,26,12);

        sprintf(buffer,"Next control in    minutes");
        display_result(buffer,45,5);
        sprintf(buffer,"%02d",next_pfc);
        display_result(buffer,61,5);

        }

/*-------------------------------------
        calculate the delta count during time t in minutes
-------------------------------------*/

double diff_cpt(VARS *diff_var, int diff_time)
        {
        int j;
        int i_samp,i_prev,nb_samp;
        double total_count;

        total_count=0;
        nb_samp=ceil(diff_time*60/TSAMP);
        for(j=0;j<nb_samp;j++)
                {
                i_samp=(diff_var->i-j)&NB_SAMP;
                i_prev=(i_samp-1)&NB_SAMP;
                total_count+= ( diff_var->val[i_samp]>=diff_var->val[i_prev]) ?
                diff_var->val[i_samp]-diff_var->val[i_prev] : diff_var->val[i_samp];
                }
        return(total_count);
        }

/*-------------------------------------
        calculate the variable variation during time t in minutes
-------------------------------------*/

double diff_var(VARS *diff_var, int diff_time)
        {
        double dvar_dt;
        int nb_samp;

        nb_samp=ceil(diff_time*60/TSAMP);
        dvar_dt=diff_var->val[diff_var->i]-diff_var->val[(diff_var->i
        -nb_samp)&NB_SAMP];
        return(dvar_dt);
        }

/*-------------------------------------
        calculate the average during time t in minutes
-------------------------------------*/

double average_var(VARS *diff_var, int diff_time)
        {
        int j;
        int i_samp,nb_samp;
        double average;

        average=0;
        nb_samp=ceil(diff_time*60/TSAMP);
        for(j=0;j<nb_samp;j++)
                {
                i_samp=(diff_var->i-j)&NB_SAMP;
                average+=diff_var->val[i_samp];
                }
        average/=nb_samp;
        return(average);
        }

/*-------------------------------------
        calculate the average^2 during time t in minutes
-------------------------------------*/

double average2_var(VARS *diff_var, int diff_time)
        {
        int j;
        int i_samp,nb_samp;
        double average;

        average=0;
        nb_samp=ceil(diff_time*60/TSAMP);
        for(j=0;j<nb_samp;j++)
                {
                i_samp=(diff_var->i-j)&NB_SAMP;
                average+=pow(diff_var->val[i_samp],2);
                }
        average/=nb_samp;
        return(average);
        }

/*-------------------------------------
        fill val[i] with the current value
-------------------------------------*/

fill_struct_var(VARS *fill_struct)

        {
        int jj;

        for(jj=0;jj<=NB_SAMP;jj++)
                {
                fill_struct->val[jj]=fill_struct->value;
                }
        }


/*-------------------------------------
        fill val[i] with the current value and delta between each value
-------------------------------------*/

fill_struct_cpt(VARS *fill_struct,double _delta)

        {
        int jj,kk,ll;

        for(jj=0;jj<NB_SAMP;jj++)
                {
                kk=(fill_struct->i-jj)&NB_SAMP;
                ll=(kk-1)&NB_SAMP;
                fill_struct->val[ll]=fill_struct->val[kk]+_delta;
                }
        }


/*-------------------------------------
        calculate the slope of variable by the least mean square method
-------------------------------------*/

double slope_var(VARS *slope_var,int diff_time)
        {
        int ii,jj,kk;
        int nb_samp;
        double slope;
        double sumxi, sumyi, sumxiyi, sumxi2;

        sumxi=0;
        sumyi=0;
        sumxiyi=0;
        sumxi2=0;

        nb_samp=ceil(diff_time*60/TSAMP);
        for(ii=0;ii<nb_samp;ii++)
                {
                jj=slope_var->i-ii;
                kk=(slope_var->i-ii)&NB_SAMP;
                sumxi+=jj;
                sumyi+=slope_var->val[kk];
                sumxiyi+=jj*slope_var->val[kk];
                sumxi2+=pow((double)jj,2);
                }
        slope=nb_samp*(nb_samp*sumxiyi-sumxi*sumyi)/(nb_samp*sumxi2-sumxi*sumxi);
        return(slope);
        }


/*-------------------------------------
        calculate the slope of counter by the least mean square method
-------------------------------------*/

double slope_cpt(VARS *slope_cpt,int diff_time)
        {
        int ii,jj,kk,ll;
        int nb_samp;
        double slope;
        double sumxi, sumyi, sumxiyi, sumxi2;
        double raz_cpt;

        raz_cpt=0;
        sumxi=0;
        sumyi=0;
        sumxiyi=0;
        sumxi2=0;

        nb_samp=ceil(diff_time*60/TSAMP);
        for(ii=0;ii<nb_samp;ii++)
```

```
                {
                jj=slope_cpt->i-ii;
                kk=(slope_cpt->i-ii)&NB_SAMP;
                ll=(kk-1)&NB_SAMP;
                sumxi+=jj;
                sumyi+=(slope_cpt->val[kk]-raz_cpt);
                sumxiyi+=jj*(slope_cpt->val[kk]-raz_cpt);
                sumxi2+=pow((double)jj,2);
                if(slope_cpt->val[ll]>slope_cpt->val[kk])
                        {
                        raz_cpt=slope_cpt->val[ll];
                        }
                }
        slope=nb_samp*(nb_samp*sumxiyi-sumxi*sumyi)/(nb_samp*sumxi2-sumxi*sumxi);
        return(slope);
        }


        /*************************************************
        Sign
        **************************************************/
         signe(double x)
        {
        x=(x<0) ? -1 : 1;
        return(x);
        }



/*---------------------------------------
        mathematical model for ADERSA
---------------------------------------*/

double adersa(double CXA, double flux, double dil, int horiz)
        {
        REACT react_adersa;
        double v, vout, prod;
        int k;
        char *s;

        react_adersa.Cno3=nitrate.value;
        react_adersa.temp=temperature.value;
        react_adersa.Fr=flux;

        v  = CXA;       /* valeur initiale de la concentration*/

        /* integration du modele toutes les minutes */
        for(k=1;k<=horiz*DT;k++)
                {
                double Delta;

                react_adersa.Cxa=v;
                model(&react_adersa,USE_FR);
                Delta=1/60.0*(react_adersa.rxa-dil*v);
                v+=Delta;

                }

                prod=v*dil*(VOLUME_TOTAL/1000.);
                return(prod);
}



/*---------------------------------------
        control programm
---------------------------------------*/

void control_spiru()
        {

        double prod, dil, delfr;
        double Fr1, Fr2, prod_ref, prod1, prod2, cxa_moy;
        double qe_max, qe_min, prod_max, prod_min;

        acq_vars();

        display_status("Control running ...");

        /* calcul de la production toutes les minutes */
        cxa_moy=average_var(&cxa,10);
        prod=cxa_moy*qe_real.value;
        production.sp=prod;

        if(!(next_pfc--))
```

```
                {
                /* algorithme de commande PFC */


                                /* contraintes sur le debit et sur la production */
                qe_real.sp=qe_nom.value;
                qe_max=qe_nom.value*(1+DQ);
                qe_min=qe_nom.value*(1-DQ);
                prod_max=qe_max*CXA_MAX;
                prod_min=qe_min*CXA_MIN;

                /* calcul de la consigne de production realisable */
                cons_prod_real.sp=max(prod_min,min(prod_max,cons_prod_nom.value));

                /* calcul du debit reel demande */
                if(cons_prod_real.sp/CXA_MAX>qe_nom.value)
                        {qe_real.sp=min(qe_max,cons_prod_nom.value/CXA_MAX);}
                if(cons_prod_real.sp/CXA_MIN<qe_nom.value)
                        {qe_real.sp=max(qe_min,cons_prod_nom.value/CXA_MIN);}
                dil=qe_real.sp*1000/VOLUME_TOTAL;

                /* trajectoire de reference */
                prod_ref=cons_prod_real.sp-pow(LAMBDA,NHC)*(cons_prod_real.sp-prod);

                /*premier scenario */
                Fr1=Fr.value;
                prod1=adersa(cxa_moy,Fr1,dil,NHC);

                /* deuxieme scenario */
                delfr=DFR*signe(cons_prod_real.sp-prod);
                Fr2=Fr1+delfr;
                prod2=adersa(cxa_moy,Fr2,dil,NHC);

                /* calcul de Fr */
                Fr.sp=Fr.value+(prod_ref-prod1)/(prod2-prod1)*delfr;

                /* contrainte sur Fr */
                Fr.sp=max(FR_MIN,min(FR_MAX,Fr.sp));

                /* calcul de la sortie modele */
                prod_mod.sp = adersa(cxa_moy,Fr.sp,dil,1);

                /* control niveau 0 - consigne lumiere */
                air_lift.Cxa=cxa_moy;
                air_lift.Cno3=nitrate.value;
                air_lift.temp=temperature.value;
                air_lift.Fr=Fr.sp;
                model(&air_lift,USE_FR);
                Eb.sp=air_lift.Eb;

                /* consigne dilution */
                act_pompe.sp=qe_real.sp/cpt_cxa_unit.value*1000./60.;

                next_pfc=DT;
                }

        send_vars();
        result();
        }
```

```
/********************************************************************
        NAME            GPSFILE.C
        AUTHOR          BINOIS C
        DESCRIPTION
                management of gps files
        UPDATES
                25-03-93
********************************************************************/

#include <malloc.h>
#include <process.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

#include "melissa.h"
#include "userdef.h"

static GPS_FILE *gps;

/*-------------------------------------
        open all gps files declared in GPS.FIL
-------------------------------------*/

GPS_FILE *open_file_gps()
        {
        FILE    *fp;
        int     hl,i,file_rank;
        char    buffer[80],str[80];
        GPS_FILE    *gp,*gp_save;
        int     file_opened;

        file_opened=OFF;
        file_rank=1;
        display_status("opening GPS files ...");
        fp=fopen("GPS.FIL","r");
        if(fp==NULL)
                {
                display_error("Could not open file GPS.FIL ... *** program stopped ***");
                exit(0);
                }
        while(fscanf(fp,"%s",buffer)!=EOF)
                {
                if(sscanf(buffer,"BASE:%s",str)!=1)
                        {
                        display_error("Syntax error in file GPS.FIL ... *** program stopped ***");
                        fclose(fp);
                        exit(0);
                        }
                for(i=1;;i++)
                        {
                        sprintf(buffer,"%s%02d.GPS",str,i);
                        hl=open_gr(buffer);
                        if(hl==-1)
                                {
                                break;
                                }
                        gp=(GPS_FILE *)malloc(sizeof(GPS_FILE));
                        if(gp==NULL)
                                {
                                sprintf(buffer,"Can't allocate memory for %s%02d.GPS *** program stopped ***",str,i);
                                display_error(buffer);
                                exit(0);
                                }
                        if(!file_opened)
                                {
                                gp->next=gp;
                                gp_save=gp;
                                }
                        sprintf(gp->file,"%s",buffer);
                        gp->handler=hl;
                        gp->rank=file_rank;
                        gp->next=gp_save->next;
                        gp_save->next=gp;
                        gp_save=gp;
                        file_opened=ON;
                        file_rank++;
                        sprintf(buffer,"file %s opened ...\n",gp->file);
                        _outtext(buffer);
                        wait_time(1);
                        }
                }
        if(!file_opened)
                {
                display_error("No GPS file found ... *** program terminated ***");
                exit(0);
                }
        fclose(fp);
        gps=gp;
        return(gp);
        }

/*-------------------------------------
        active one group using GPS_FILE struct
-------------------------------------*/

GPS_FILE *activ_grp_gps()

        {
        GPS_FILE    *gp;
        char    buffer[80];
        int     ret_activ_gr;

        gp=gps->next;
        ret_activ_gr=activ_gr(gp->handler);
        if(ret_activ_gr==-1)
                {
                sprintf(buffer,"Can't activate file %s ... *** program stopped ***",gps->file);
                display_error(buffer);
                exit(0);
                }
        gps=gp;
        display_activ_group(gps);
        return(gp);
        }


/*-------------------------------------
        close all groups using GPS_FILE struct
-------------------------------------*/

void close_grp_gps()

        {
        GPS_FILE    *gp;
        char    buffer[80];
        int     file_rank, ret_close, all_closed;

        file_rank=gps->rank;
        all_closed=OFF;
        while(!all_closed)
                {
                ret_close=close_gr(gps->handler);
                if(ret_close==-1)
                        {
                        sprintf(buffer,"\nCan't close file %s ...\n",gps->file);
                        display_error(buffer);
                        error_gps();
                        }
                else
                        {
                        sprintf(buffer,"\nFile %s closed",gps->file);
                        _outtext(buffer);
                        wait_time(1);
                        }
                gps=gps->next;
                if(gps->rank==file_rank)
                        {
                        all_closed=ON;
                        }
                }
        display_no_activ_group();
        }
```

```
/*********************************************************************
        NAME            MELFCT.C
        AUTHOR          BINOIS C
        DESCRIPTION
                general functions listing file
        UPDATES
                10-03-93
*********************************************************************/

#include <graph.h>
#include <time.h>
#include <stdio.h>
#include <stdlib.h>
#include <process.h>
#include <string.h>
#include <malloc.h>
#include <math.h>

#include "melissa.h"
#include "userdef.h"

/*--------------------------------------
        time base generator
--------------------------------------*/

timebase()
        {
        static  unsigned long lastsamp = 0; /* last sampling */
        static  unsigned tsamp = TSAMP ;    /* sampling interval */
        static  unsigned long last_display;
        time_t  ltime;
        char    buffer[100];
        struct  tm  *dt;
        double  minute;

        time(&ltime);
        if(last_display<ltime)
                {
                dt=localtime(&ltime);
                dt->tm_mon++;
                sprintf(buffer," %02d/%02d/%02d   %02d:%02d:%02d",dt->tm_mday,
                dt->tm_mon,dt->tm_year,dt->tm_hour,dt->tm_min,dt->tm_sec);
                display_time(buffer);
                last_display=ltime;
                }

        time(&ltime);
        if (lastsamp==0)
                {
                display_status("timebase synchronisation ...");
                minute=TSAMP/60;
                while((ceil(dt->tm_min/minute)!=dt->tm_min/minute)||(dt->tm_sec!=0))
                        {
                        time(&ltime);
                        dt=localtime(&ltime);
                        dt->tm_mon++;
                        if(last_display<ltime)
                                {
                                sprintf(buffer," %02d/%02d/%02d   %02d:%02d:%02d",dt->tm_mday,
                                dt->tm_mon,dt->tm_year,dt->tm_hour,dt->tm_min,dt->tm_sec);
                                display_time(buffer);
                                last_display=ltime;
                                }
                        }
                lastsamp=ltime;
                display_status(" ");
                return(0);
                }
        if ((lastsamp+tsamp)<=ltime)
                {
                lastsamp=lastsamp+tsamp;
                return(0);
                }
        return(ltime);
        }


/*--------------------------------------
        wait i seconds
--------------------------------------*/

wait_time(int i)
        {
        char    buffer[100];
        unsigned long start_time;

        while(timebase()==0)
                {
                }
        start_time=timebase();
        while(timebase()<start_time+i)
                {
                }
        }
```

```
/*******************************************************************

NAME            SCREEN.C

AUTHOR          BINOIS C

DESCRIPTION
        screen and graphic functions listing file

UPDATES
        06-05-93

*******************************************************************/


#include <stdio.h>
#include <graph.h>
#include "melissa.h"


/*-------------------------------------
        screen initialisation
-------------------------------------*/

void screen_init(void)
        {
        _setvideomode(_TEXTC80);
        _setbkcolor((long)RED);
        _settextwindow(1,1,25,80);
        _clearscreen(_GWINDOW);

        _settextcolor(BLACK);
        tab(3,19);
        _outtext("messages ...");

        _setbkcolor((long)BLUE);
        _settextcolor(WHITE);
        tab(26,2);
        _outtext("  ***  M E L I S S A  ***  ");

        _settextwindow(4,2,18,79);
        _clearscreen(_GWINDOW);
        _settextwindow(20,2,24,79);
        _clearscreen(_GWINDOW);

        use_message_window();
        _wrapon(_GWRAPON);
        _displaycursor(_GCURSOROFF);

        }

/*-------------------------------------
        display activ group
-------------------------------------*/

display_activ_group(GPS_FILE *gps)
        {
        char    buffer[100];
        struct  rccoord txtpos;
        txtpos=_gettextposition();

        use_group_window();
        sprintf(buffer,"ACTIVE GROUP : %s",gps->file);
        _outtext(buffer);

        use_message_window();
        _settextposition(txtpos.row,txtpos.col);
        }

/*-------------------------------------
        display no activ group
-------------------------------------*/

display_no_activ_group()
        {
        char    buffer[100];
        struct  rccoord txtpos;
        txtpos=_gettextposition();

        use_group_window();
        sprintf(buffer,"ACTIVE GROUP : --------.GPS");
        _outtext(buffer);

        use_message_window();
        _settextposition(txtpos.row,txtpos.col);
        }
```

```
/*-------------------------------------
        display current time
-------------------------------------*/

display_time(char *buffer)
        {
        struct  rccoord txtpos;
        txtpos=_gettextposition();

        use_time_window();
        _outtext(buffer);

        use_message_window();
        _settextposition(txtpos.row,txtpos.col);
        }

/*-------------------------------------
        display result in main window
-------------------------------------*/

display_result(char *buffer,short x,short y)
        {
        struct  rccoord txtpos;

        if((x>76)||(y>12))
                {
                display_error("Can't display result : coordinates error on :");
                display_error(buffer);
                return(-1);
                }
        txtpos=_gettextposition();

        use_display_window();
        _settextposition(y,x);
        _outtext(buffer);

        use_message_window();
        _settextposition(txtpos.row,txtpos.col);
        }


/*-------------------------------------
        display system status
-------------------------------------*/

display_status(char *buffer)
        {
        struct  rccoord txtpos;
        txtpos=_gettextposition();

        use_status_window();
        _outtext(buffer);

        use_message_window();
        _settextposition(txtpos.row,txtpos.col);
        }

/*-------------------------------------
        display error messages
-------------------------------------*/

display_error(char *buffer)
        {
        _setbkcolor((long)GREEN);
        _settextcolor(RED+16);
        _outtext(buffer);
        printf("\a\a\a");
        _setbkcolor((long)BLUE);
        _settextcolor(WHITE);
        _wrapon(_GWRAPON);
        _displaycursor(_GCURSOROFF);
        _outtext("\n");
        }

/*-------------------------------------
        use messages area
-------------------------------------*/

use_message_window()
        {
        _settextwindow(20,3,24,78);
        _setbkcolor((long)BLUE);
        _settextcolor(WHITE);
        _wrapon(_GWRAPON);
        _displaycursor(_GCURSOROFF);
        }
```

```
/*-------------------------------------
        use group display area
-------------------------------------*/

use_group_window()
        {
        _settextwindow(4,3,4,39);
        tab(3,4);
        _settextcolor(WHITE);
        _setbkcolor((long)BLUE);
        _clearscreen(_GWINDOW);
        }

/*-------------------------------------
        use time display area
-------------------------------------*/

use_time_window()
        {
        _settextwindow(4,58,4,79);
        tab(58,4);
        _settextcolor(WHITE);
        _setbkcolor((long)BLUE);
        }

/*-------------------------------------
        use status display area
-------------------------------------*/

use_status_window()
        {
        _settextwindow(18,3,18,78);
        tab(3,18);
        _settextcolor(WHITE);
        _setbkcolor((long)BLUE);
        _clearscreen(_GWINDOW);
        }

/*-------------------------------------
        use main display area
-------------------------------------*/

use_display_window()
        {
        _settextwindow(6,3,17,78);
        _settextcolor(WHITE);
        _setbkcolor((long)BLUE);
        tab(3,6);
        }

/*-------------------------------------
        move the cursor to the position (x,y)
-------------------------------------*/

tab(short x,short y)
        {
        _settextposition(y,x);
        }
```

```
/*********************************************************************

NAME            SPIRULIN.C

AUTHOR          BINOIS C

DESCRIPTION
        MAIN PROGRAM listing file

UPDATES
        03-05-93

*********************************************************************/

#include <graph.h>
#include <process.h>
#include <stdio.h>
#include <signal.h>
#include <stdlib.h>

#include "userdef.h"
#include "melissa.h"

int     my_interrupt();
char    buffer[100];
GPS_FILE *gps, *open_file_gps(), *activ_grp_gps();
int pointer;
VARS    essai;

main()
        {
        void wait_time(int);
        char    chr;
        int flag;

/*---------------------------------------
        screen initialisation
---------------------------------------*/

        screen_init();

/*---------------------------------------
        set interruption
---------------------------------------*/

        if(signal(SIGINT,my_interrupt)==(int(*)())-1)
                {
                _outtext("\nCouldn't set SIGINT *** Program Terminated ***\n");
                exit(0);
                }


/*---------------------------------------
        timebase synchronisation
---------------------------------------*/

        wait_time(1);

/*---------------------------------------
        open groups and active one
---------------------------------------*/

        gps=open_file_gps();
        gps=activ_grp_gps();
        set_gps(gps);

/*---------------------------------------
        variables initialisation
---------------------------------------*/

        init_vars();
        init_alarm();

/*---------------------------------------
        waiting loop
---------------------------------------*/

        do
                {
                if(!timebase())
                        {
                        control_spiru();
                        flag=0;
                        }
                else
                        {
                        if(!flag)
                                {
```

```
                                alarm_spiru();
                                check_network();
                                flag=1;
                                }
                        }
                if(kbhit())
                        {
                        chr=getch();
                        }
                }
        while(1);
        }

/*-------------------------------------
        interruption of programm
-------------------------------------*/

int     my_interrupt()
        {
        char    ch;
        signal(SIGINT,SIG_IGN);
        _outtext("Terminate processing ? ");
        ch=getch();
        if((ch=='y')||(ch=='Y'))
                {
                close_grp_gps();
                _outtext("\nbye.... *** Program Terminated by user ***\n");
                wait_time(5);
                _setvideomode(_DEFAULTMODE);
                exit(0);
                }
        if(signal(SIGINT,my_interrupt)==(int(*)())-1)
                {
                _outtext("\nCouldn't set SIGINT *** Program Terminated ***\n");
                exit(0);
                }
        _outtext("Continue...\n");
        return;
        }
```

```
/*******************************************************************

        NAME            VARS.C

        AUTHOR          BINOIS C

        DESCRIPTION
                access to data via gps functions
                and variables management

        UPDATES
                03-05-93

*******************************************************************/


#include <time.h>
#include <stdio.h>
#include <stdlib.h>
#include <process.h>
#include <string.h>
#include <malloc.h>

#include "melissa.h"
#include "userdef.h"

static GPS_FILE *gps;
GPS_FILE *activ_grp_gps();
int my_interrupt();

/*-------------------------------------
        acquisition of gps pointer
-------------------------------------*/

void set_gps(GPS_FILE *gp)
        {
        gps=gp;
        }


/*-------------------------------------
        read function
-------------------------------------*/

void read_var(VARS *read_var)

        {
        double  value;
        double  read_gps(VARS *);
        int     file_rank;
        char    buffer[80];

        value=read_gps(read_var);
        if(value==-1)
                {
                file_rank=gps->rank;
                while(value==-1)
                        {
                        gps=activ_grp_gps();
                        value=read_gps(read_var);
                        if(file_rank==gps->rank)
                                {
                                sprintf(buffer,"Can not find %s in gps files",read_var->name);
                                display_error(buffer);
                                my_interrupt();
                                }
                        }
                }
        read_var->i++;
        read_var->i&=NB_SAMP;
        read_var->val[read_var->i]=value;
        read_var->value=value;
        }

/*-------------------------------------
        read gps sub_function
-------------------------------------*/

double read_gps(VARS *read_vars)

        {
        OGPS    read_ogps;
        IGPS    read_igps;
        int     read_count,j,k;
        int     ret_code,tag_found;
        char    buffer[9],name[9];
        double  read_value;
```

```
read_value=0;
for(read_count=0;read_count<1;read_count++)
{
switch(read_vars->tag_cmd)
        {
        case TAG:
                {
                ret_code=rd_gps(read_vars->name,&read_igps);
                if(ret_code==-1)
                        {
                        error_gps();
                        return(-1);
                        }
                switch(ret_code)
                        {
                        case ISBIT:
                                {
                                if(read_igps.i.d.val_state==ACTIV)
                                        {read_value=1;}
                                else
                                        {read_value=0;}
                                if(!read_vars->update)
                                        {
                                        if(read_igps.i.d.act_state==ACTIV)
                                                {read_vars->max=1;}
                                        else
                                                {read_vars->max=0;}
                                        read_vars->dev_num=read_igps.h.dev_num;
                                        sprintf(read_vars->file,"%s",gps->file);
                                        read_vars->update=ON;
                                        }
                                break;
                                }
                        case ISABUS:
                                {
                                read_value=read_igps.i.bus.val;
                                if(!read_vars->update)
                                        {
                                        read_vars->min=read_igps.i.bus.l;
                                        read_vars->max=read_igps.i.bus.h;
                                        sprintf(read_vars->unit,"%s",read_igps.i.bus.unit);
                                        read_vars->dev_num=read_igps.h.dev_num;
                                        sprintf(read_vars->file,"%s",gps->file);
                                        read_vars->update=ON;
                                        }
                                break;
                                }
                        }
                break;
                }
        case CMD:
                {
                ret_code=set_cmd(read_vars->name,&read_ogps);
                if(ret_code==-1)
                        {
                        error_gps();
                        return(-1);
                        }
                switch(ret_code)
                        {
                        case OSBIT:
                        case OSREGIST:
                                {
                                sprintf(buffer,"%s is not a valid tag name for MICON ..."
                                ,read_vars->name);
                                display_error(buffer);
                                break;
                                }
                        case OSMILOOP:
                                {
                                read_value=read_ogps.o.ml.val;
                                read_vars->out=read_ogps.o.ml.out;
                                read_vars->sp=read_ogps.o.ml.sp;
                                if(!read_vars->update)
                                        {
                                        read_vars->min=read_ogps.o.ml.spmin;
                                        read_vars->max=read_ogps.o.ml.spmax;
                                        sprintf(read_vars->unit,"%s",read_ogps.o.ml.spunit);
                                        read_vars->dev_num=read_ogps.h.dev_num;
                                        sprintf(read_vars->file,"%s",gps->file);
                                        read_vars->update=ON;
                                        }
                                break;
                                }
                        case OSMILOC:
                                {
                                read_value=read_ogps.o.loc.val;
                                read_vars->sp=read_ogps.o.loc.val;
```

```
                                        if(!read_vars->update)
                                                {
                                                read_vars->min=read_ogps.o.loc.locmin;
                                                read_vars->max=read_ogps.o.loc.locmax;
                                                sprintf(read_vars->unit,"%s",read_ogps.o.loc.locunit);
                                                read_vars->dev_num=read_ogps.h.dev_num;
                                                sprintf(read_vars->file,"%s",gps->file);
                                                read_vars->update=ON;
                                                }
                                        break;
                                        }
                                case OSMIVD:
                                        {
                                        read_value=read_ogps.o.vd.val;
                                        read_vars->sp=read_ogps.o.vd.val;
                                        if(!read_vars->update)
                                                {
                                                read_vars->dev_num=read_ogps.h.dev_num;
                                                sprintf(read_vars->file,"%s",gps->file);
                                                read_vars->update=ON;
                                                }
                                        break;
                                        }
                                }
                        break;
                        }
                default:
                        {
                        tag_found=OFF;
                        j=nbtags+nbcommands;
                        for(k=0;k<j;k++)
                                {
                                sprintf(buffer,"%s",gettags(k));
                                if(strcmp(buffer,read_vars->name)==0)
                                        {
                                        tag_found=ON;
                                        if(k<nbtags)
                                                {
                                                ret_code=rd_gps(read_vars->name,&read_igps);
                                                read_vars->tag_cmd=TAG;
                                                }
                                        else
                                                {
                                                ret_code=set_cmd(read_vars->name,&read_ogps);
                                                read_vars->tag_cmd=CMD;
                                                }
                                        if(ret_code==-1)
                                                {
                                                error_gps();
                                                return(-1);
                                                }
                                        else
                                                {
                                                read_vars->type=ret_code;
                                                read_count--;
                                                }
                                        break;
                                        }
                                }
                        if(!tag_found)
                                {
                                return(-1);
                                }
                        }
                }
        }
        if(read_value==-1)
                {
                return(-0.9999999);
                }
        return(read_value);
        }

/*-------------------------------------
        write function
-------------------------------------*/

void
write_var(VARS *write_var)

        {
        int     file_rank,ret_code;
        char    buffer[80];

        ret_code=write_gps(write_var);
        if(ret_code==-1)
                {
                file_rank=gps->rank;
                while(ret_code==-1)
                        {
                        gps=activ_grp_gps();
                        ret_code=write_gps(write_var);
                        if(file_rank==gps->rank)
                                {
                                sprintf(buffer,"Can not find %s in gps files",write_var->name);
                                display_error(buffer);
                                my_interrupt();
                                break;
                                }
                        }
                }
        }


/*---------------------------------------
        write gps sub_function
---------------------------------------*/

int write_gps(VARS *write_vars)

        {
        OGPS    write_ogps;
        S_USER  write_s_user;
        int     ret_code,tag_written;
        time_t  ltime;
        char    buffer[80];

        tag_written=OFF;
        while(!tag_written)
        {
        switch(write_vars->tag_cmd)
                {
                case TAG:
                        {
                        sprintf(buffer,"Can't write the TAG %s ...",write_vars->name);
                        display_error(buffer);
                        my_interrupt();
                        break;
                        }
                case CMD:
                        {
                        ret_code=set_cmd(write_vars->name,&write_ogps);
                        if(ret_code==-1)
                                {
                                error_gps();
                                return(-1);
                                }
                        switch(ret_code)
                                {
                                case OSBIT:
                                case OSREGIST:
                                        {
                                        sprintf(buffer,"%s is not a valid tag name for MICON ..."
                                        ,write_vars->name);
                                        display_error(buffer);
                                        break;
                                        }
                                case OSMILOOP:
                                        {
                                        if((write_ogps.o.ml.state&16)==16)
                                                {
                                                write_s_user.mic.mode=MICLOCREM;
                                                write_s_user.mic.status_sta=ACTIV;
                                                if(wr_gps(&write_ogps,&write_s_user)==-1)
                                                        {
                                                        error_gps();
                                                        }
                                                break;
                                                }
                                        if((write_ogps.o.ml.state&1)==0)
                                                {
                                                write_s_user.mic.mode=MICAUTO;
                                                write_s_user.mic.status_sta=ACTIV;
                                                if(wr_gps(&write_ogps,&write_s_user)==-1)
                                                        {
                                                        error_gps();
                                                        }
                                                break;
                                                }
                                        write_s_user.mic.status_sta=INACTIV;
                                        write_s_user.mic.status_out=INACTIV;
                                        write_s_user.mic.status_ra=INACTIV;
                                        write_s_user.mic.status_bi=INACTIV;
                                        write_s_user.mic.status_loc=INACTIV;
                                        write_s_user.mic.status_vd=INACTIV;
                                        write_s_user.mic.val_sp=write_vars->sp;
                                        write_s_user.mic.status_sp=ACTIV;
                                        tag_written=ON;
                                        break;
                                        }
                                case OSMILOC:
                                        {
                                        write_s_user.mic.val_loc=write_vars->sp;
                                        write_s_user.mic.status_loc=ACTIV;
                                        tag_written=ON;
                                        break;
                                        }
                                case OSMIVD:
                                        {
                                        write_s_user.mic.val_vd=(int)write_vars->sp;
                                        write_s_user.mic.status_vd=ACTIV;
                                        tag_written=ON;
                                        break;
                                        }
                                }
                        break;
                        }
                default:
                        {
                        read_var(write_vars);
                        }
                }
        }
        if(wr_gps(&write_ogps,&write_s_user)==-1)
                {
                error_gps();
                time(&ltime);
                sprintf(buffer,"When writing %s at time %s",write_vars->name,
                ctime(&ltime));
                display_error(buffer);
                return(0);
                }
        else
                {
                return(0);
                }
        }

/*-------------------------------------
        errors management
-------------------------------------*/

error_gps()
        {
        switch (gpserror)
                {
                case ERDOS:
                        {
                        display_error("A DOS problem has occured ...\n");
                        my_interrupt();
                        break;
                        }
                case ENETW:
                        {
                        display_error("Network or Mailbox fault ...\n");
                        break;
                        }
                case INVALID_FGPS:
                        {
                        display_error("Incorrect GPS file ...\n");
                        my_interrupt();
                        break;
                        }
                case ETAGCMD:
                        {
                        break;
                        }
                case ETAGTYP:
                case ECMDTYP:
                        {
                        display_error("Unknown CMD or TAG ...");
                        my_interrupt();
                        break;
                        }
                case ECONV:
                        {
                        display_error("Floating point conversion error:ignored ...");
                    /* my_interrupt();*/
                        break;
                        }
                case EEQUIP:
                        {
                        display_error("Unknown PLC protocol ...");
                        my_interrupt();
                        break;
                        }
                case ENUMPLC:
                        {
                        display_error("Invalid device number ...");
                        my_interrupt();
                        break;
                        }
                default:
                        {
                        display_error("Unidentified error ...");
                        my_interrupt();
                        break;
                        }
                }
        }

/*------------------------------------
        check if mailbox is refresh
------------------------------------*/

check_network()
        {
        display_status("Checking Network ...");
        while(garde(101,1))
                {
                }
        display_status(" ");
        }
```

```
/***********************************************************************

NAME            USERDEF.H

AUTHORS         (C) TOPTOOLS 1988,1989,1990

DESCRIPTION

        INDUSTAR General Purpose Station - User Include File

UPDATES
        90-05-10 - Add TiWay support

***********************************************************************/


/* ---------------------------
        Miscellaneous constants definition
--------------------------- */

#define ACTIV           1                   /* command is active                                */
#define INACTIV         0                   /* command is inactive                              */

#define ON                      1                   /* ON  value  for digital commands          */
#define OFF                     0                   /* OFF value  "      "      "               */


/* ---------------------------
        rd_gps() return codes
--------------------------- */

                                                        /* Tag type                          Structure
/
#define ISBIT           1                   /* digital                                  IDIG    */
#define ISABUS          2                   /* analog                                   IBUSA   */


/* ---------------------------
        set_cmd() return codes
--------------------------- */

                                                        /* Cmd type                          Structure
/
#define OSBIT           32                  /* digital                                  ODIG    */
#define OSMILOOP        41                  /* analog  (Micon loop)             OMLO    */
#define OSREGUL         42                  /* analog  (AB,TCS,PLS loop)    OREGU   */
#define OSREGIST        52                  /* analog  (register)               OREGIS  */
#define OSMIVD          61                  /* digital (Micon DV)               OMVD    */
#define OSMILOC         71                  /* analog  (Micon loc)              OMLOC   */


/* ---------------------------
        gpserror values
--------------------------- */

#define ERDOS           1                   /* DOS problem                                      */
#define ENETW           2                   /* network or mailbox                       */
#define INVALID_FGPS 3              /* invalid GPS file                                 */
#define ETAGCMD         4                   /* tag or command not found                 */
#define EVARCOD         5                   /* invalid variable codification        */
#define ECONV           6                   /* floating point conversion            */
#define ETAGTYP         7                   /* unknown tag type                                 */
#define EEQUIP          8                   /* unknown PLC protocol                     */
#define ECMDTYP         9                   /* unknown command type                     */
#define ENUMPLC         10                  /* invalid device number                    */
#define ERPROTECT       11                  /* protection key not found                 */
#define ENSECTOR        12                  /* invalid record number                    */
#define ETYPVARCAL      13                  /* not a computed variable                  */
#define EGDPREV         24                  /* action on previous GD var not over   */
#define EGDTYPVAR       25                  /* not a GD variable                        */
#define EGDPOLLIS       26                  /* POLLIS.TAB not found                     */
#define EGDCMDFAIL      27                  /* failed command to GD                     */
#define ELOCAL          31                  /* PLC in Local state (cmd impossible)  */
#define ISMODAUTO       32                  /* AUTO mode : change is impossible         */
#define EPLSOVER        44                  /* PLStar variable value out of limits  */


/* ---------------------------
        Not selected command fields
--------------------------- */

#define FVALNOTUSED     0xF4240         /* 4bytes, not selected analog cmd field*/
#define IVALNOTUSED 64                  /* 1byte,  not selected dig or loop fld.*/


/* ---------------------------
        Specific definitions for Micon equipments
--------------------------- */
```

```
#define MICLOCREM       151                     /* local/remote                                                 */
#define MICCASCA        150                     /* cascade                                                              */
#define MICAUTO         149                     /* auto                                                                 */
#define MICMANUAL       148                     /* manual                                                               */
#define ISINCONFIG      155                     /* Micon is starting configuration                 */


/* ---------------------------
        External declarations
--------------------------- */

extern int gpserror;                    /* variable to house error codes               */
extern int echonetw ;                   /* network error code, when applicable  */
extern unsigned int nbtags;             /* # tags in activated group                   */
extern unsigned int nbcommands; /* # cmds in activated group                   */
extern char *gettags () ;               /* tag or command name (8-char string)  */


/* ----------------------------------------------------------------------

                                                  STRUCTURES FOR READING TAGS

---------------------------------------------------------------------- */


/* ---------------------------
        Common header structure (tags and commands)
--------------------------- */

typedef struct _iohead  {

        char      tag[9];                        /* tag or cmd                                                  */
        char      tag_name[31];          /*  "       "  name                                       */
        unsigned  char tag_type;         /* tag or cmd type :
                                                                        = 1  digital input
                                                                        = 2  analog   "
                                                                        = 3  digital output
                                                                        = 4  analog     "                (loop)
                                                                        = 5    "        "                (register)
                                                                        = 6    "        "                Micon  VD
                                                                        = 7    "        "                Micon  LOC
                                                                        ..................................    */

        unsigned char   zone;            /* INDUSTAR C&C Station number                        */
        unsigned int    dev_num;         /* controller number                                          */
        unsigned char   dev_type;        /* controller type :
                                                                        = 1  MICON
                                                                        = 2  JBUS-MODBUS
                                                                        = 3  STRUTHERS & DUNN
                                                                        = 4  ALLEN BRADLEY
                                                                        = 5  UNITELWAY
                                                                        = 8  TCS 6000
                                                                        = 9  TIWAY
                                                                        = 13 LAC
                                                                        = 14 PLStar TT
                                                                        = 15 Ghost Device TT
                                                                        ..................................    */

        unsigned char   log_can;         /*  logical channel number                                    */

} IOHEAD;


/* ---------------------------
        Digital Input
--------------------------- */

typedef struct _idig {

        unsigned char val_state;                /* tag state  (0/1)                                     */
        unsigned char act_state;                /* active state                                         */
        char              alarm_tag[9];         /* associated alarm tag                         */
        char              fault_tag[9];         /* associated fault tag                         */
        char              progress_tag[9];      /* associated in progress tag           */

        char              var_nam[7] ;          /* TIWAY - process variable name        */
        int                       var_typ ;                     /* TIWAY - process variable type       */
        int                       var_num ;                     /* TIWAY - process variable #          */

} IDIG;


/* ---------------------------
        Analog tag
--------------------------- */

typedef struct _ibusa {
        float             val;                          /* value  (IEEE floating point)        */
        float             scale;                        /* scale                                                       */
        float             vl;                           /* very low limit                                      */
        float             l;                            /* low limit                                           */
        float             h;                            /* high limit                                          */
        float             vh;                           /* very high limit                                     */
        char              unit[5];                      /* unit                                                        */

        char              var_nam[7] ;          /* TIWAY - process variable name       */
        int                       var_typ ;                    /* TIWAY - process variable type        */
        int                       var_num ;                    /* TIWAY - process variable #           */

} IBUSA;


/* ---------------------------
        Digital or Analog tag (without the header structure)
--------------------------- */

typedef union _u_inp {

        IDIG    d;                                              /* when digital                                        */
        IBUSA   bus;                                    /* when analog                                         */

} U_INP;


/* ---------------------------
        Structure for the calls to rd_gps()
--------------------------- */

typedef struct _igps {

        IOHEAD  h;                                              /* common header structure                     */
        U_INP   i;                                              /* according to tag type and PLC       */

} IGPS;


/* ----------------------------------------------------------------------

                                   STRUTURES FOR GETTING COMMAND INFORMATION

---------------------------------------------------------------------- */


/* ---------------------------
        Digital command
--------------------------- */

typedef struct _odig {

        char    cmd_tag[9];                     /* tag name associated to the command   */
        char    st_cmd;                         /* tag  state (0/1)                                    */
        char    as_cmd;                         /* active state                                        */
        char    cmd_file;                       /* file number for ALLEN BRADLEY       */
        int             cmd_typ ;                       /* TIWAY
/

        char    plcloc_tag[9];          /* local/remote tag                                    */
        char    st_plcloc;                      /* local(0)/remote(1) tag state        */
        char    as_plcloc;                      /* active state                                        */

        char    alarm_tag[9];           /* alarm tag associated to the command */
        char    st_alarm;                       /* state alarm tag (0/1)                       */
        char    as_alarm;                       /* active state                                        */

        char    fault_tag[9];           /* fault tag                                           */
        char    st_fault;                       /* state fault tag (0/1)                       */
        char    as_fault;                       /* active state                                        */

        char    instart_tag[9];         /* in progress tag                                     */
        char    st_instart;                     /* state in progress tag (0/1)         */
        char    as_instart;                     /* active state                                        */

        char    inhalt_tag[9];          /* in halt tag                                         */
        char    st_inhalt;                      /* state in halt tag (0/1)                     */
        char    as_inhalt;                      /* active state                                        */

        char    localcmd_tag[9];        /* local command tag (element)         */
        char    st_localcmd;            /* state local command tag (0/1)       */
        char    as_localcmd;            /* active state                                        */

        char    cmdtype;                        /* command type = 0,1,2,3                      */
        char    mult_tab;                       /* not used by the G.P.S.                      */

        unsigned devon_adr;                     /* PLC bit state adresse ON                    */
```

```
       char     as_devon;                  /* active state for ON                                */
       char     mask_on;                   /* bit mask for ALLEN BRADLEY                  */

       unsigned devoff_adr;         /* PLC bit state addresse OFF                   */
       char     as_devoff;                 /* active state for OFF                               */
       char     mask_off;                  /* bit mask for ALLEN BRADLEY                  */

} ODIG;


/* ---------------------------
       Analog command (register)
--------------------------- */

typedef struct _oregis {

       char     cmd_tag[9];         /* tag name associated to the command  */
       float    val;                       /* tag measure value                                  */
       int             cmd_typ ;                  /* TIWAY
/

       unsigned char cmd_mod;       /* mode variable PLStar                                */
       char     locrem_tag[9];      /* tag for local/remote state                  */
       char     state;                     /* PLC state Local(0)/Remote(1)                */
       char     as_locrem;                 /* active state Local/Remote                   */

       char     reg1_tag[9];        /* tag register 1                                     */
       float    reg1_val;                  /* value register 1                                          */
       float    reg1_min;                  /* value mini  reg. 1                                 */
       float    reg1_max;                  /* value maxi  reg. 1                                 */
       char     reg1_unit[5];       /* unit register 1                                    */
       unsigned reg1_adrhx;         /* PLC adress (hexa) of register 1              */
       unsigned char reg1_mod;      /* mode variable PLStar                                */
       char     reg1_file;                 /* # file AB                                                 */
       int             reg1_typ ;                 /* TIWAY
/
       int             reg1_num ;                 /* TIWAY
/

       char     reg2_tag[9];        /* tag register 2                                     */
       float    reg2_val;                  /* value register 2                                          */
       float    reg2_min;                  /* value mini  reg. 2                                 */
       float    reg2_max;                  /* value maxi  reg. 2                                 */
       char     reg2_unit[5];       /* unit reg. 2                                        */
       unsigned reg2_adrhx;         /* PLC adress (hexa) of register 2              */
       unsigned char reg2_mod;      /* mode variable PLStar                                */
       char     reg2_file;                 /* # file AB                                                 */
       int             reg2_typ ;                 /* TIWAY
/
       int             reg2_num ;                 /* TIWAY
/

       char     reg3_tag[9];        /* tag register 3                                     */
       float    reg3_val;                  /* value register 3                                          */
       float    reg3_min;                  /* valeur mini reg. 3                                 */
       float    reg3_max;                  /* valeur maxi reg. 3                                 */
       char     reg3_unit[5];       /* unit register 3                                    */
       unsigned reg3_adrhx;         /* PLC adress (hexa) of register 3              */
       unsigned char reg3_mod;      /* mode variable PLStar                                */
       char     reg3_file;                 /* # file AB                                                 */
       int             reg3_typ ;                 /* TIWAY
/
       int             reg3_num ;                 /* TIWAY
/

} OREGIS;


/* ---------------------------
       Analog command (loop)
--------------------------- */

typedef struct _oregu {

       char     cmd_tag[9];         /* tag name associated to the command  */
       float    val;                       /* tag measure value                                  */
       int             cmd_typ ;                  /* TIWAY
/
       int             tiloopnb ;                 /* TIWAY
/
       char     cmd_mod;                   /* mode variable PLStar                               */

       char     spt_tag[9];         /* tag for setpoint                                   */
       float    spt_val;                   /* setpoint value                                            */
       float    spt_min;                   /* value mini  spt                                           */
       float    spt_max;                   /* value maxi  spt                                           */
       char     spt_unit[5];        /* unit for setpoint                                   */
       unsigned spt_adr;                   /* adresse ALLEN B                                           */
```

```
        unsigned spt_file;                   /*  file number of spt value (A-B)              */
        char     spt_mod;                    /* mode variable PLStar (cf. PLStar)    */
        int               spt_typ ;                   /* TIWAY
/
        int               spt_num ;                   /* TIWAY
/


        char     mo_tag[9];                  /* tag for Manual Output                                */
        float    mo_val;                     /* M.O. value                                                      */
        float    mo_min;                     /* value mini  M.O.                                                */
        float    mo_max;                     /* value maxi  M.O.                                                */
        char     mo_unit[5];         /* unit  M.O.                                            */            */
        unsigned mo_adr;                     /* adresse ALLEN B
        unsigned mo_file;                    /* file number of M.O.                                  */
        char     mo_mod;                     /* mode variable PLStar (cf.PLStar)             */
        int               mo_typ ;                    /* TIWAY
/
        int               mo_num ;                    /* TIWAY
/

        char     stalo_tag[9];       /* tag for loop status                                */
        char     stalo_mod;                  /* status loop value                                 */
        char     stalo_unit[5];      /* unit for loop status                               */
        unsigned stalo_adr;                  /* adresse ALLEN B                                                 */
        unsigned stalo_file;         /* file number of loop status                */
        char     stat_mod;                   /* mode variable PLStar (cf. PLStar)    */
        int               stalo_typ ;                 /* TIWAY                                                                  */
        int               stalo_num ;                 /* TIWAY                                                                  */

} OREGU;


/* ----------------------------
        Analog command (Micon loop)
---------------------------- */


typedef struct _omlo {

        char     regutag[9];         /* tag associated to the command               */
        float    val;                        /* current value read in the mailbox     */                        */
        int               nloop;                      /* loop number                                          */
        float    sp;                         /* setpoint value
        float    spmin;                      /*   "     minimum value                               */
        float    spmax;                      /*   "     maximum  "                                  */
        char     spunit[5];                  /*  unit for setpoint                                  */
        float    out;                        /* outpoint value                                                   */
        float    outmin;                     /*   "     minimum value                               */
        float    outmax;                     /*   "     maximum   "                                 */
        char     outunit[5];         /* unit for outpoint                                          */
        float    ratio;                      /* ratio value                                                      */
        float    ratiomin;                   /*   "     mini                                                     */
        float    ratiomax;                   /*   "     maxi                                                     */
        char     ratiounit[5];       /*  unit for ratio                                             */
        float    bias;                       /* bias value                                                       */
        float    biasmin;                    /*   "     mini                                                     */
        float    biasmax;                    /*   "     maxi                                                     */
        char     biasunit[5];        /*   unit for bias                                                */
        unsigned char state;         /* loop state (auto/manual/cascade)               */

 } OMLO;


 /* ----------------------------
        Analog command (Micon LOC)
 ---------------------------- */

 typedef struct _omloc {

        char     loctag[9];                  /* tag name associated to the command    */
        float    val;                        /* current LOC value                                */
        int               nloc;                       /* LOC number                                                    */
        float    locmin;                     /*  minimum LOC value                              */
        float    locmax;                     /*  maximum  "   "                                            */
        char     locunit[5];         /*  unit for LOC                                                  */

 } OMLOC;


 /* ----------------------------
        Digital command (Micon DV)
 ---------------------------- */

 typedef struct _omvd {

        char     vdtag[9];                   /* tag name associated to the command    */
        int               val;                        /* current Discrete Virtual value                 */
        int              nvd;                           /* Discrete Virtual number                                 */

} OMVD;


/* ---------------------------
        Digital or Analog command (without the header)
--------------------------- */

typedef union _u_outp {

        ODIG    d;                                      /* digital (common for all PLCs)                   */
        OMLO    ml;                                     /* Loop: MICON                                                     */
        OREGU   l;                                      /* Loop: ALLEN-BRADLEY, PLStar, TIWAY   */
        OREGIS  r;                                      /* Register: MODBUS, JBUS, LAC, PLStar */
                                                                /* ...ALLEN-B, UNITELWAY, TIWAY                */
        OMVD    vd;                                     /* MICON VD
/
        OMLOC   loc;                            /* MICON LOC                                               */

} U_OUTP;


/* ---------------------------
        Command structure for the calls to set_cmd() and wr_gps()
--------------------------- */

typedef struct _ogps {

        IOHEAD  h;                                      /* common header structure                                 */
        U_OUTP  o;                                      /* according to the PLC                                    */

} OGPS;


/* -----------------------------------------------------------------------

                                                  STRUCTURES TO SEND COMMANDS

----------------------------------------------------------------------- */


/* ---------------------------
        Sending a command to JBUS, MODBUS, LAC equipments
--------------------------- */

typedef struct _usjbus {

        char    status_bit;                     /* cmd state switch ACTIV/INACTIV (default)     */
        char    action_bit;                     /* new bit value (ON / OFF)                        */

        char    status_reg1;            /* cmd reg.1 switch ACTIV/INACTIV (default)     */
        float   newreg1;                        /* new  register 1 value                           */

        char    status_reg2;            /* cmd reg. 2 switch ACTIV/INACTIV      */
        float   newreg2;                        /* new  register 2 value                           */

        char    status_reg3;            /* cmd reg. 3 switch ACTIV/INACTIV      */
        float   newreg3;                        /* new register 3 value                            */

} USJBUS;


/* ---------------------------
        Send a command to a Micon equipment
--------------------------- */

typedef struct _usmloop {

        char     status_sp;                     /* cmd setpoint switch ACTIV/INACTIV(default)   */
        float    val_sp;                        /* new setpoint value                              */

        char     status_out;            /* cmd outpoint switch ACTIV/INACTIV    */
        float    val_out;                       /* new outpoint value                              */

        char     status_ra;                     /* cmd ratio switch ACTIV/INACTIV               */
        float    val_ra;                        /* new ratio value                                         */

        char     status_bi;                     /* cmd bias switch ACTIV/INACTIV                */
        float    val_bi;                        /* new bias value                                          */

        char     status_sta;            /* cmd state switch ACTIV/INACTIV               */
        unsigned char mode;                     /* new channel state value - use above MICON
                                                                   definitions (MICLOCREM etc.)            */

        char     status_loc;            /* cmd LOC switch ACTIV/INACTIV                 */
        float    val_loc;                       /* new LOC value                                           */
        char      status_vd;               /* cmd DV switch ACTIV/INACTIV             */
        char      val_vd;                  /* new DV value  ON/OFF                           */

} USMLOOP;


/* ---------------------------
        Send a command to :
        Allen-Bradley, PLStar, UnitelWay, Reliance, TCS or TIWAY equipment
--------------------------- */

typedef struct _usalb {

        char      status_sp1;       /* cmd setpoint or register 1 switch ACTIV/INACTIV(default)     */
        float     val_sp1;                  /* new setpoint or register 1 value         */

        char      status_out2;      /* cmd outpoint or register 2 switch ACTIV/INACTIV      */
        float     val_out2;                 /* new outpoint or register 2 value         */

        char      status_sta3;      /* cmd loop state or register 3 switch ACTIV/INACTIV    */
        float     val_sta3;                 /* new loop state value or register 3   */

        char      status_bit;       /* cmd state switch ACTIV/INACTIV              */
        char      val_bit;                  /* new bit value  ON/OFF                          */

} USALB;


/* ---------------------------
        Send a command to the Mailbox
--------------------------- */

typedef struct _uscalc {

        char      status_bit;       /* state switch (ACTIV/INACTIV) for new bit value       */
        char      val_bit;                  /* new bit value                                          */

        char      status_num;       /* state switch (ACTIV/INACTIV) for new analog value    */
        float     val_num;                  /* new analog value                                       */

        char      destable[32];     /* list of zones to send the message    */

} USCALC;


/* ---------------------------
        Structure for the calls to wr_gps()
--------------------------- */

typedef union _s_user {

        USJBUS  jb;             /* MODBUS,JBUS,LAC                                                    */
        USMLOOP mic;    /* MICON                                                                      */
        USALB   ab;             /* ALLEN BRADLEY,PLStar,UNITELWAY,TCS,RELIANCE,TIWAY*/
        USCALC  cal;    /* Ghost Device TT                                                    */

} S_USER;


/* ---------------------------
        GPS functions declarations
--------------------------- */

extern  int open_gr(char *name_gr);
extern  int activ_gr(int h_gr);
extern  int rd_gps(char *mtag,struct _igps *_igps);
extern  int set_cmd(char *nom,struct _ogps *ogps);
extern  int wr_gps(struct _ogps *stout,union _s_user *stuser);
extern  int close_gr(int h_gr);
extern  char *gettags(unsigned int ii);
extern  int garde(unsigned int n,unsigned int nsecs);
```

```
/*****************************************************************

        NAME            MELISSA.H

        AUTHOR          BINOIS C

        DESCRIPTION
                General Declarations

        UPDATES
                02-06-93

*****************************************************************/

/*-------------------------------------
        constants for VARS
-------------------------------------*/
#define TAG     128
#define CMD     255
#define UNDEF   0
#define NB_SAMP 0xFF          /* number of samples stored in val[ ] */


/*-------------------------------------
        structure for variables
-------------------------------------*/

typedef struct _vars {
        char    name[9];        /* tag name             */
        char    file[12];       /* file name            */
        int     type;           /* rd_gps/set_cmd return code                   */
        unsigned char tag_cmd;  /* TAG or CMD           */
        unsigned int dev_num;   /* controller number    */
        double  value;          /* current value        */
        int     i;              /* pointer on last value entered in val[] */
        double  val[NB_SAMP+1]; /* previous values      */
        double  min;
        double  max;
        double  sp;             /* set point for LOOP       */
        double  out;            /* out value for LOOP       */
        char    unit[5];        /* unit for analog values  */
        char    update;         /* ON when structure updated*/
} VARS;

/*-------------------------------------
        structure for opened GPS files
-------------------------------------*/

typedef struct _gps_file{
        char    file[15];           /* file name                */
        int     handler;            /* handler of gps file      */
        int     rank;               /* rank of the gps file     */
        struct  _gps_file   *next;  /* next opened gps file     */
} GPS_FILE;


/*-------------------------------------
        structure for reactor state
-------------------------------------*/

typedef struct  _react{
        double  Cxa;
        double  Cno3;
        double  temp;
        double  press;
        double  Eb;
        double  Fr;
        double  rxa;
        double  rn;
        double  ro2;
} REACT;

/*-------------------------------------
        general constants
-------------------------------------*/

#define TSAMP   60      /* sampling interval in secondes    */
#define SYNCHRO  1
#define ERROR_SPEED 0.01 /* max error for the model          */
#define VOLUME_LIGHT 3900
#define VOLUME_TOTAL 7000
#define CPT_CXA_UNIT 41.68

/*-------------------------------------
        Model control constants
-------------------------------------*/

#define USE_FR  10
```

```
#define USE_EB  20


/*-------------------------------------
        Mathematical constants
-------------------------------------*/

#define PI      3.14159265359


/*-------------------------------------
        colours
-------------------------------------*/

#define BLACK   0
#define BLUE    1
#define GREEN   2
#define CYAN    3
#define RED     4
#define MAGENTA 5
#define BROWN   6
#define WHITE   7

/*-------------------------------------
        ADERSA constants
-------------------------------------*/

#define DT      30  /* periode de commande en minutes */
#define NHC     5       /* coincidence point in DT        */
#define LAMBDA  0.75    /* reference trajectory dynamic   */
#define DFR 10.  /* radiant flux increment W/m2    */
#define FR_MIN  10.     /* min constraint on Fr W/m2      */
#define FR_MAX  8000.    /* max constraint on Fr W/m2      */
#define DQ              0.1     /* flow variation                 */
#define CXA_MIN 500     /* min constaint on cxa mg/l      */
#define CXA_MAX 1500    /* max constaint on cxa mg/l      */
```

Annex D :

C code file with non linear predictive control

(V2.1)

```
/********************************************************************

        NAME            ALARMS.C

        AUTHOR          BINOIS C

        DESCRIPTION
                ALARM management listing file

        UPDATES
                05-05-93

********************************************************************/

#include <math.h>
#include <stdio.h>
#include <stdlib.h>

#include "userdef.h"
#include "melissa.h"

/*--------------------------------------
        variables declarations
-------------------------------------*/


char buffer[100];

/*--------------------------------------
        alarm programm
-------------------------------------*/

void alarm_spiru()
        {
        acq_alarm();
        send_alarm();
        }

/*--------------------------------------
        alarms initialisation
-------------------------------------*/

init_alarm()
        {
        display_status("Initialisation of ALARMS ...");
        wait_time(2);
        display_status(" ");
        }

/*--------------------------------------
        alarms acquisition
-------------------------------------*/

acq_alarm()
        {
        display_status("Acquisition of ALARMS ...");

        wait_time(2);
        display_status(" ");
        }

/*--------------------------------------
        alarms updating
-------------------------------------*/

send_alarm()
        {
        display_status("Updating ALARMS ...");

        wait_time(2);
        display_status(" ");
        }
```

```
/*******************************************************************

        NAME            CONTROL.C

        AUTHOR          BINOIS C    (modified by FULGET N. ADERSA)

        DESCRIPTION
                CONTROL PROGRAM listing file

        UPDATES
                20-09-95

*******************************************************************/

#include <malloc.h>
#include <math.h>
#include <stdio.h>
#include <stdlib.h>

#include "userdef.h"
#include "melissa.h"

int my_interrupt();

/*--------------------------------------
        variables declarations
--------------------------------------*/

VARS    cxa;            /* biomass concentration          */
VARS    nitrate;        /* nitrate concentration          */
VARS    cal_nitrate;    /* nitrate calibration switch     */

VARS    Eb;             /* light intensity in the reactor */
VARS    Fr;             /* incident flux                  */
VARS    temperature;    /* temperature in the reactor     */
VARS    pH;             /* pH of culture                  */
VARS    act_pompe;      /* dilution pump action           */
VARS    cal_pump;       /* calibration of pump (l/h)      */

/********************   variables ADERSA   *****************/

VARS    cons_prod_nom;  /* nominal production setpoint    */
VARS    cons_prod_real; /* feasible production setpoint   */
VARS    qe_nom;         /* nominal flow setpoint          */
VARS    qe_real;        /* feasible flow setpoint         */
VARS    production;     /* measured production            */
VARS    prod_mod;       /* model production               */

int     next_pfc;       /* next execution of PFC          */
char    buffer[100];

/*--------------------------------------
        mathematical model
--------------------------------------*/

#ifndef ADERSA
model(REACT *react, int mode)
#else
model(react, mode)
REACT *react;
int mode;
#endif
   {

        double  zpc=.135;
        double  zp=.57;
        double  zch=0.0085;
        double  zg=0.;
        double  za;
        double  Ea=871.;
        double  Es=167.;
        double  alpha,delta,delta3;
        double  Fr,Fr1,Fr2,Fr3;
        double  R,R1,R2,R3,Rb;
        double  z;
        double  jstep=0.01;
        double  pij,pijz;
        double  Kj=20;
        double  KN=5.3;
        double  muM=0.54;
        double  yn=0.42;

        double  coeft,coefN,Rmean;
        R=0.048;
        R1=0.0302;
        R2=0.02585;
        R3=0.0115;
        Rb=0.0095;

/*  general parameters ----------------------------------*/

        za=zpc+zch;
        alpha=sqrt(za*Ea/(za*Ea+(1+zg)*Es));
        delta=(za*Ea+(1+zg)*Es)*react->Cxa/1000.*alpha*R;
        delta3=delta*R2/R;

        switch(mode) {

                case USE_EB:
                {
                /*  incident flux determination --------------*/
                Fr3=react->Eb*Rb/PI/R3;
                z=R3/R2;
                Fr2=Fr3*z/(2*alpha)*(cosh(delta3)+alpha*sinh(delta3))/sinh(delta3*z);
                Fr1=Fr2*R2/R1;
                z=R1/R;
                Fr=Fr1*z/(2*alpha)*(cosh(delta)+alpha*sinh(delta))/sinh(delta*z);
                react->Fr=Fr;
                break;
                }

                case USE_FR:
                {
                /*  Eb determination -------------------------*/
                z=R1/R;
                Fr=react->Fr;
                Fr1=Fr/(z/(2*alpha)*(cosh(delta)+alpha*sinh(delta))/sinh(delta*z));
                Fr2=Fr1/(R2/R1);
                z=R3/R2;
                Fr3=Fr2/(z/(2*alpha)*(cosh(delta3)+alpha*sinh(delta3))/sinh(delta3*z));
                react->Eb=Fr3/(Rb/PI/R3);
                break;
                }

                default:
                {
                display_error("Incorrect model call ...\n");
                display_error("*** Program terminated ***\n");
                exit(0);
                }
        }

/*  determination of the mean growth rate ---------------*/

        pij=0;
        for(z=jstep/2;z<=1-jstep/2;z+=jstep)
                {
                if((z<R2/R)||(z>R1/R))
                        {
                        pijz=Fr/z*2*cosh(delta*z)/(cosh(delta)+alpha*sinh(delta));
                        if(pijz>=1)
                                {
                                pij+=2*z*pijz/(Kj+pijz)*jstep;
                                }
                        }
                }
        Rmean=muM*pij*zpc*react->Cxa*VOLUME_LIGHT/VOLUME_TOTAL;

/* temperature and nitrates correction */

        coeft=0.8*exp(-pow((react->temp-35)/10,2))+0.2;
        coefN=react->Cno3/(KN+react->Cno3);


/******  nitrate saturation   ********/
coefN=1;
/************************************/
        react->rxa=Rmean*coefN*coeft;
        react->rn=yn*react->rxa;


        }

/*--------------------------------------
        copy structure reacta to reactb
--------------------------------------*/

#ifndef ADERSA
copy_react(REACT *reacta,REACT *reactb)
#else
copy_react(reacta,reactb)
REACT *reacta;
REACT *reactb;
#endif
```

```
        {
        reactb->Cxa=reacta->Cxa;
        reactb->Cno3=reacta->Cno3;
        reactb->temp=reacta->temp;
        reactb->press=reacta->press;

        reactb->Eb=reacta->Eb;
        reactb->Fr=reacta->Fr;
        reactb->rxa=reacta->rxa;
        reactb->rn=reacta->rn;
        reactb->ro2=reacta->ro2;
        }


/*-------------------------------------
        variables initialisation
-------------------------------------*/

init_vars()
        {
        REACT   init_react;
        double  delta;
        int     jj;

        display_status("Initialisation of variables ...");

/*  TAG and COMMAND name initialisation     */

        sprintf(cxa.name,"LOOP0107");
        sprintf(nitrate.name,"LOOP0103");
        sprintf(cal_nitrate.name,"DI--0125");

        sprintf(Eb.name,"LOOP0105");
        sprintf(Fr.name,"LOC-0128");
        sprintf(temperature.name,"LOOP0106");
        sprintf(pH.name,"LOOP0104");
        sprintf(act_pompe.name,"LOC-0154");
        sprintf(cal_pump.name,"LOC-0137");

        sprintf(cons_prod_nom.name,"LOC-0150");
        sprintf(cons_prod_real.name,"LOC-0152");
        sprintf(qe_nom.name,"LOC-0151");
        sprintf(qe_real.name,"LOC-0153");
        sprintf(production.name,"LOC-0155");
        sprintf(prod_mod.name,"LOC-0156");

/*  Variables initialisation    */

        acq_vars();

        init_react.Cxa=cxa.value;
        init_react.Cno3=nitrate.value;
        init_react.temp=temperature.value;
        init_react.Eb=Eb.value;
        model(&init_react,USE_EB);
        Fr.sp=init_react.Fr;
        write_var(&Fr);
        fill_struct_var(&cxa);

        cons_prod_real.sp=cons_prod_nom.value;
        write_var(&cons_prod_real);
        qe_real.sp=qe_nom.value;
        write_var(&qe_real);

/* initialisation timer PFC */
        next_pfc=DT;

        wait_time(1);

        display_status(" ");
        }

/*-------------------------------------
        variables acquisition
-------------------------------------*/

acq_vars()
        {
        display_status("Acquisition of variables ...");

        read_var(&cxa);

/*  nitrate analyser calibration  */

        read_var(&cal_nitrate);
        if(!cal_nitrate.value)
                {
```

```
                read_var(&nitrate);
                }

        read_var(&Eb);
        read_var(&Fr);
        read_var(&temperature);
        read_var(&pH);
        read_var(&act_pompe);
        read_var(&cal_pump);

        read_var(&cons_prod_nom);
        read_var(&cons_prod_real);
        read_var(&qe_nom);
        read_var(&qe_real);
        read_var(&production);
        read_var(&prod_mod);

        display_status(" ");
        }

/*---------------------------------------
        commands updating
-------------------------------------*/

send_vars()
        {
        display_status("Updating variables ...");

        write_var(&Eb);
        write_var(&Fr);
        write_var(&act_pompe);
        write_var(&cons_prod_real);
        write_var(&qe_real);
        write_var(&production);
        write_var(&prod_mod);

        display_status(" ");
        }

/*---------------------------------------
        prepare result of control for display
-------------------------------------*/

result()
        {
#ifndef ADERSA
        void display_result(char *,short,short);
#else
        void display_result();
#endif
        char buffer[150];

        sprintf(buffer,"Concentrations  Biomass");
        display_result(buffer,1,1);
        sprintf(buffer,"mg/l");
        display_result(buffer,32,1);
        display_result(buffer,32,2);
        sprintf(buffer,"Nitrate");
        display_result(buffer,17,2);
        sprintf(buffer,"%.1f",cxa.value);
        display_result(buffer,26,1);
        sprintf(buffer,"%.1f",nitrate.value);
        display_result(buffer,26,2);

        sprintf(buffer,"Light");
        display_result(buffer,1,4);
        sprintf(buffer,"Eb        W/m2");
        display_result(buffer,22,4);
        sprintf(buffer,"Fr        W/m2");
        display_result(buffer,22,5);
        sprintf(buffer,"%.1f",Eb.sp);
        display_result(buffer,26,4);
        sprintf(buffer,"%.1f",Fr.sp);
        display_result(buffer,26,5);

        sprintf(buffer,"Production     measured        mg/h");
        display_result(buffer,1,7);
        sprintf(buffer,"%.2f",production.sp);
        display_result(buffer,26,7);

        sprintf(buffer,"set-point        mg/h");
        display_result(buffer,15,8);
        sprintf(buffer,"%.2f",cons_prod_nom.value);
        display_result(buffer,26,8);

        sprintf(buffer,"realised        mg/h");
        display_result(buffer,16,9);
        sprintf(buffer,"%.2f",cons_prod_real.sp);
        display_result(buffer,26,9);

        sprintf(buffer,"model         mg/h");
        display_result(buffer,15,10);
        sprintf(buffer,"%.2f",prod_mod.sp);
        display_result(buffer,26,10);

        sprintf(buffer,"Flow            realised        l/h");
        display_result(buffer,1,11);
        sprintf(buffer,"%.3f",qe_real.sp);
        display_result(buffer,26,11);

        sprintf(buffer,"set point        l/h");
        display_result(buffer,15,12);
        sprintf(buffer,"%.3f",qe_real.value);
        display_result(buffer,26,12);

        sprintf(buffer,"Next control in    minutes");
        display_result(buffer,45,5);
        sprintf(buffer,"%02d",next_pfc);
        display_result(buffer,61,5);

        }


/*--------------------------------------
        calculate the delta count during time t in minutes
-------------------------------------*/

#ifndef ADERSA
double diff_cpt(VARS *diff_var, int diff_time)
#else
double diff_cpt(diff_var, diff_time)
VARS *diff_var;
int diff_time;
#endif
        {
        int j;
        int i_samp,i_prev,nb_samp;
        double total_count;

        total_count=0;
        nb_samp=ceil(diff_time*60/TSAMP);
        for(j=0;j<nb_samp;j++)
                {
                i_samp=(diff_var->i-j)&NB_SAMP;
                i_prev=(i_samp-1)&NB_SAMP;
                total_count+= ( diff_var->val[i_samp]>=diff_var->val[i_prev]) ?
                diff_var->val[i_samp]-diff_var->val[i_prev] : diff_var->val[i_samp];
                }
        return(total_count);
        }


/*--------------------------------------
        calculate the variable variation during time t in minutes
-------------------------------------*/

#ifndef ADERSA
double diff_var(VARS *diff_var, int diff_time)
#else
double diff_var(diff_var, diff_time)
VARS *diff_var;
int diff_time;
#endif
        {
        double dvar_dt;
        int nb_samp;

        nb_samp=ceil(diff_time*60/TSAMP);
        dvar_dt=diff_var->val[diff_var->i]-diff_var->val[(diff_var->i
        -nb_samp)&NB_SAMP];
        return(dvar_dt);
        }

/*--------------------------------------
        calculate the average during time t in minutes
-------------------------------------*/

#ifndef ADERSA
double average_var(VARS *diff_var, int diff_time)
#else
double average_var(diff_var, diff_time)
VARS *diff_var;
int diff_time;
#endif
        {
        int j;
        int i_samp,nb_samp;
        double average;

        average=0;
        nb_samp=ceil(diff_time*60/TSAMP);
        for(j=0;j<nb_samp;j++)
                {
                i_samp=(diff_var->i-j)&NB_SAMP;
                average+=diff_var->val[i_samp];
                }
        average/=nb_samp;
        return(average);
        }

/*-------------------------------------
        calculate the average^2 during time t in minutes
-------------------------------------*/

#ifndef ADERSA
double average2_var(VARS *diff_var, int diff_time)
#else
double average2_var(diff_var, diff_time)
VARS *diff_var;
int diff_time;
#endif
        {
        int j;
        int i_samp,nb_samp;
        double average;

        average=0;
        nb_samp=ceil(diff_time*60/TSAMP);
        for(j=0;j<nb_samp;j++)
                {
                i_samp=(diff_var->i-j)&NB_SAMP;
                average+=pow(diff_var->val[i_samp],2);
                }
        average/=nb_samp;
        return(average);
        }

/*-------------------------------------
        fill val[i] with the current value
-------------------------------------*/

#ifndef ADERSA
fill_struct_var(VARS *fill_struct)
#else
fill_struct_var(fill_struct)
VARS *fill_struct;
#endif

        {
        int jj;

        for(jj=0;jj<=NB_SAMP;jj++)
                {
                fill_struct->val[jj]=fill_struct->value;
                }
        }


/*-------------------------------------
        fill val[i] with the current value and delta between each value
-------------------------------------*/

#ifndef ADERSA
fill_struct_cpt(VARS *fill_struct,double _delta)
#else
fill_struct_cpt(fill_struct,_delta)
VARS *fill_struct;
double _delta;
#endif

        {
        int jj,kk,ll;

        for(jj=0;jj<NB_SAMP;jj++)
                {
                kk=(fill_struct->i-jj)&NB_SAMP;
                ll=(kk-1)&NB_SAMP;
                fill_struct->val[ll]=fill_struct->val[kk]+_delta;
                }
        }


/*-------------------------------------
        calculate the slope of variable by the least mean square method
```

```
------------------------------------*/

#ifndef ADERSA
double slope_var(VARS *slope_var,int diff_time)
#else
double slope_var(slope_var,diff_time)
VARS *slope_var;
int diff_time;
#endif
        {
        int ii,jj,kk;
        int nb_samp;
        double slope;
        double sumxi, sumyi, sumxiyi, sumxi2;

        sumxi=0;
        sumyi=0;
        sumxiyi=0;
        sumxi2=0;

        nb_samp=ceil(diff_time*60/TSAMP);
        for(ii=0;ii<nb_samp;ii++)
                {
                jj=slope_var->i-ii;
                kk=(slope_var->i-ii)&NB_SAMP;
                sumxi+=jj;
                sumyi+=slope_var->val[kk];
                sumxiyi+=jj*slope_var->val[kk];
                sumxi2+=pow((double)jj,2);
                }
        slope=nb_samp*(nb_samp*sumxiyi-sumxi*sumyi)/(nb_samp*sumxi2-sumxi*sumxi);
        return(slope);
        }


/*------------------------------------
        calculate the slope of counter by the least mean square method
------------------------------------*/

#ifndef ADERSA
double slope_cpt(VARS *slope_cpt,int diff_time)
#else
double slope_cpt(slope_cpt,diff_time)
VARS *slope_cpt;
int diff_time;
#endif
        {
        int ii,jj,kk,ll;
        int nb_samp;
        double slope;
        double sumxi, sumyi, sumxiyi, sumxi2;
        double raz_cpt;

        raz_cpt=0;
        sumxi=0;
        sumyi=0;
        sumxiyi=0;
        sumxi2=0;

        nb_samp=ceil(diff_time*60/TSAMP);
        for(ii=0;ii<nb_samp;ii++)
                {
                jj=slope_cpt->i-ii;
                kk=(slope_cpt->i-ii)&NB_SAMP;
                ll=(kk-1)&NB_SAMP;
                sumxi+=jj;
                sumyi+=(slope_cpt->val[kk]-raz_cpt);
                sumxiyi+=jj*(slope_cpt->val[kk]-raz_cpt);
                sumxi2+=pow((double)jj,2);
                if(slope_cpt->val[ll]>slope_cpt->val[kk])
                        {
                        raz_cpt=slope_cpt->val[ll];
                        }
                }
        slope=nb_samp*(nb_samp*sumxiyi-sumxi*sumyi)/(nb_samp*sumxi2-sumxi*sumxi);
        return(slope);
        }



        /*************************************************
        Sign
        *************************************************/
#ifndef ADERSA
 signe(double x)
#else
 signe(x)
double x;
```

```
#endif
        {
        x=(x<0) ? -1 : 1;
        return(x);
        }



/*-------------------------------------
        mathematical model for ADERSA
-------------------------------------*/

#ifndef ADERSA
double predimod(REACT react, double dil, int horiz)
#else
double predimod(react, dil, horiz)
REACT react;
double dil;
int horiz;
#endif
        {
/* react: current reactor model state   (Cxa,Fr,Cno3,temp)      */
/* dil  : dilution rate                 (in h-1)                */
/* horiz: prediction horizon            (in number of DT)       */
/* prod : predicted production          (in mg/h)               */

        double v, prod;
        int k;

        v  = react.Cxa;      /* current biomass concentration */

        /* model integration (sampling period 1mn) */
        for(k=1;k<=horiz*DT;k++)
                {
                double Delta;

                react.Cxa=v;
                model(&react,USE_FR);
                Delta=1/60.0*(react.rxa-dil*v);
                v+=Delta;

                }

                prod=v*dil*VOLUME_TOTAL;
                return(prod);
}



/*--------------------------------------
        control programm
--------------------------------------*/

void control_spiru()
        {

        double  Fr1, Fr2, delfr;                                /*in W/m2 */
        double  prod_ref,prod1,prod2,prod_max,prod_min;         /*in mg/h */
        double  qe_max, qe_min;                                 /*in l/h */
        double  dil;                                            /*in h-1 */
        double  cxa_moy , nit_moy;                              /*in mg/l */
        REACT   react;
        acq_vars();

        display_status("Control running ...");

        if(!(next_pfc--))
                {
                /* control PFC algorithm */

                /* biomass concentration */
                cxa_moy=average_var(&cxa,10);
                nit_moy=average_var(&nitrate,10);

                /* production calculation  */
                production.sp=cxa_moy*qe_real.value;

                /* reactor state */
                react.Cno3=nit_moy;
                react.temp=temperature.value;
                react.Cxa=cxa_moy;

                /* flow and production constraints*/
                qe_max=qe_nom.value*(1+DQ);
                qe_min=qe_nom.value*(1-DQ);
                prod_max=qe_max*CXA_MAX;
        prod_min=qe_min*CXA_MIN;

        /* feasible production setpoint calculation*/
        cons_prod_real.sp=max(prod_min,min(prod_max,cons_prod_nom.value));

        /* real flow setpoint  and corresponding dilution rate*/
        qe_real.sp=qe_nom.value;
        if(cons_prod_real.sp/CXA_MAX>qe_nom.value)
                {qe_real.sp=min(qe_max,cons_prod_nom.value/CXA_MAX);}
        if(cons_prod_real.sp/CXA_MIN<qe_nom.value)
                {qe_real.sp=max(qe_min,cons_prod_nom.value/CXA_MIN);}
        dil=qe_real.sp/VOLUME_TOTAL;

        /* reference trajectory */
        prod_ref=cons_prod_real.sp-pow(LAMBDA,NHC)*(cons_prod_real.sp-production.sp);

        /*first scenario */
        Fr1=Fr.value;
        react.Fr = Fr1;
        prod1=predimod(react,dil,NHC);

        /* second scenario */
        delfr=DFR*signe(cons_prod_real.sp-production.sp);
        Fr2=Fr1+delfr;
        react.Fr = Fr2;
        prod2=predimod(react,dil,NHC);

        /* Fr calculation */
        Fr.sp=Fr.value+(prod_ref-prod1)/(prod2-prod1)*delfr;

        /* constraint on Fr */
        Fr.sp=max(FR_MIN,min(FR_MAX,Fr.sp));

        /* light setpoint sended to P100 controller */
        react.Fr=Fr.sp;
        model(&react,USE_FR);
        Eb.sp=react.Eb;

        /* pump setpoint sended to P100 controller */
        act_pompe.sp=qe_real.sp/cal_pump.value;

        /* model output calculation */
        prod_mod.sp = predimod(react,dil,1);

        next_pfc=DT;
        }

    send_vars();
    result();
    }
```

```
/*******************************************************************
        NAME            GPSFILE.C
        AUTHOR          BINOIS C
        DESCRIPTION
                management of gps files
        UPDATES
                25-03-93
*******************************************************************/

#ifndef ADERSA
#include <process.h>
#endif
#include <malloc.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

#include "melissa.h"
#include "userdef.h"

static GPS_FILE *gps;

/*-------------------------------------
        open all gps files declared in GPS.FIL
-------------------------------------*/

GPS_FILE *open_file_gps()
        {
        FILE    *fp;
        int     hl,i,file_rank;
        char    buffer[80],str[80];
        GPS_FILE    *gp,*gp_save;
        int     file_opened;

        file_opened=OFF;
        file_rank=1;
        display_status("opening GPS files ...");
        fp=fopen("GPS.FIL","r");
        if(fp==NULL)
                {
                display_error("Could not open file GPS.FIL ... *** program stopped ***");
                exit(0);
                }
        while(fscanf(fp,"%s",buffer)!=EOF)
                {
                if(sscanf(buffer,"BASE:%s",str)!=1)
                        {
                        display_error("Syntax error in file GPS.FIL ... *** program stopped ***");
                        fclose(fp);
                        exit(0);
                        }
                for(i=1;;i++)
                        {
                        sprintf(buffer,"%s%02d.GPS",str,i);
                        hl=open_gr(buffer);
                        if(hl==-1)
                                {
                                break;
                                }
                        gp=(GPS_FILE *)malloc(sizeof(GPS_FILE));
                        if(gp==NULL)
                                {
                                sprintf(buffer,"Can't allocate memory for %s%02d.GPS *** program stopped ***",str,i);
                                display_error(buffer);
                                exit(0);
                                }
                        if(!file_opened)
                                {
                                gp->next=gp;
                                gp_save=gp;
                                }
                        sprintf(gp->file,"%s",buffer);
                        gp->handler=hl;
                        gp->rank=file_rank;
                        gp->next=gp_save->next;
                        gp_save->next=gp;
                        gp_save=gp;
                        file_opened=ON;
                        file_rank++;
                        sprintf(buffer,"file %s opened ...\n",gp->file);
                        _outtext(buffer);
                        wait_time(1);
                        }
                }
        if(!file_opened)
                {
                display_error("No GPS file found ... *** program terminated ***");
                exit(0);
                }
        fclose(fp);
        gps=gp;
        return(gp);
        }

/*-------------------------------------
        active one group using GPS_FILE struct
-------------------------------------*/

GPS_FILE *activ_grp_gps()

        {
        GPS_FILE    *gp;
        char    buffer[80];
        int     ret_activ_gr;

        gp=gps->next;
        ret_activ_gr=activ_gr(gp->handler);
        if(ret_activ_gr==-1)
                {
                sprintf(buffer,"Can't activate file %s ... *** program stopped ***",gps->file);
                display_error(buffer);
                exit(0);
                }
        gps=gp;
        display_activ_group(gps);
        return(gp);
        }


/*-------------------------------------
        close all groups using GPS_FILE struct
-------------------------------------*/

void close_grp_gps()

        {
        GPS_FILE    *gp;
        char    buffer[80];
        int     file_rank, ret_close, all_closed;

        file_rank=gps->rank;
        all_closed=OFF;
        while(!all_closed)
                {
                ret_close=close_gr(gps->handler);
                if(ret_close==-1)
                        {
                        sprintf(buffer,"\nCan't close file %s ...\n",gps->file);
                        display_error(buffer);
                        error_gps();
                        }
                else
                        {
                        sprintf(buffer,"\nFile %s closed",gps->file);
                        _outtext(buffer);
                        wait_time(1);
                        }
                gps=gps->next;
                if(gps->rank==file_rank)
                        {
                        all_closed=ON;
                        }
                }
        display_no_activ_group();
        }
```

```
/********************************************************************
        NAME            MELFCT.C
        AUTHOR          BINOIS C
        DESCRIPTION
                general functions listing file
        UPDATES
                10-03-93
*******************************************************************/

#ifndef ADERSA
#include <graph.h>
#include <process.h>
#endif
#include <time.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <malloc.h>
#include <math.h>

#include "melissa.h"
#include "userdef.h"

/*--------------------------------------
        time base generator
--------------------------------------*/

timebase()
        {
        static  unsigned long lastsamp = 0; /* last sampling */
        static  unsigned tsamp = TSAMP ;    /* sampling interval */
        static  unsigned long last_display;
        time_t  ltime;
        char    buffer[100];
        struct  tm  *dt;
        double  minute;

        time(&ltime);
        if(last_display<ltime)
                {
                dt=localtime(&ltime);
                dt->tm_mon++;
                sprintf(buffer," %02d/%02d/%02d   %02d:%02d:%02d",dt->tm_mday,
                dt->tm_mon,dt->tm_year,dt->tm_hour,dt->tm_min,dt->tm_sec);
                display_time(buffer);
                last_display=ltime;
                }

        time(&ltime);
        if (lastsamp==0)
                {
                display_status("timebase synchronisation ...");
                minute=TSAMP/60;
                while((ceil(dt->tm_min/minute)!=dt->tm_min/minute)||(dt->tm_sec!=0))
                        {
                        time(&ltime);
                        dt=localtime(&ltime);
                        dt->tm_mon++;
                        if(last_display<ltime)
                                {
                                sprintf(buffer," %02d/%02d/%02d   %02d:%02d:%02d",dt->tm_mday,
                                dt->tm_mon,dt->tm_year,dt->tm_hour,dt->tm_min,dt->tm_sec);
                                display_time(buffer);
                                last_display=ltime;
                                }
                        }
                lastsamp=ltime;
                display_status(" ");
                return(0);
                }
        if ((lastsamp+tsamp)<=ltime)
                {
                lastsamp=lastsamp+tsamp;
                return(0);
                }
        return(ltime);
        }


/*--------------------------------------
        wait i seconds
--------------------------------------*/

#ifndef ADERSA
wait_time(int i)
#else
wait_time( i)
int i;
```

```
#endif
        {
        char    buffer[100];
        unsigned long start_time;

        while(timebase()==0)
                {
                }
        start_time=timebase();
        while(timebase()<start_time+i)
                {
                }
        }
```

```
/*********************************************************************

NAME            SCREEN.C

AUTHOR          BINOIS C

DESCRIPTION
        screen and graphic functions listing file

UPDATES
        06-05-93

*********************************************************************/


#include <stdio.h>
#include "melissa.h"


/*--------------------------------------
        screen initialisation
--------------------------------------*/

void screen_init(void)
        {
        _setvideomode(_TEXTC80);
        _setbkcolor((long)RED);
        _settextwindow(1,1,25,80);
        _clearscreen(_GWINDOW);

        _settextcolor(BLACK);
        tab(3,19);
        _outtext("messages ...");

        _setbkcolor((long)BLUE);
        _settextcolor(WHITE);
        tab(26,2);
        _outtext("  ***  M E L I S S A  ***  ");

        _settextwindow(4,2,18,79);
        _clearscreen(_GWINDOW);
        _settextwindow(20,2,24,79);
        _clearscreen(_GWINDOW);

        use_message_window();
        _wrapon(_GWRAPON);
        _displaycursor(_GCURSOROFF);

        }

/*--------------------------------------
        display activ group
--------------------------------------*/

display_activ_group(GPS_FILE *gps)
        {
        char    buffer[100];
        struct  rccoord txtpos;
        txtpos=_gettextposition();

        use_group_window();
        sprintf(buffer,"ACTIVE GROUP : %s",gps->file);
        _outtext(buffer);

        use_message_window();
        _settextposition(txtpos.row,txtpos.col);
        }

/*--------------------------------------
        display no activ group
--------------------------------------*/

display_no_activ_group()
        {
        char    buffer[100];
        struct  rccoord txtpos;
        txtpos=_gettextposition();

        use_group_window();
        sprintf(buffer,"ACTIVE GROUP : --------.GPS");
        _outtext(buffer);

        use_message_window();
        _settextposition(txtpos.row,txtpos.col);
        }


/*--------------------------------------
```

```
        display current time
-------------------------------------*/

display_time(char *buffer)
        {
        struct  rccoord txtpos;
        txtpos=_gettextposition();

        use_time_window();
        _outtext(buffer);

        use_message_window();
        _settextposition(txtpos.row,txtpos.col);
        }

/*------------------------------------
        display result in main window
-------------------------------------*/

display_result(char *buffer,short x,short y)
        {
        struct  rccoord txtpos;

        if((x>76)||(y>12))
                {
                display_error("Can't display result : coordinates error on :");
                display_error(buffer);
                return(-1);
                }
        txtpos=_gettextposition();

        use_display_window();
        _settextposition(y,x);
        _outtext(buffer);

        use_message_window();
        _settextposition(txtpos.row,txtpos.col);
        }


/*------------------------------------
        display system status
-------------------------------------*/

display_status(char *buffer)
        {
        struct  rccoord txtpos;
        txtpos=_gettextposition();

        use_status_window();
        _outtext(buffer);

        use_message_window();
        _settextposition(txtpos.row,txtpos.col);
        }


/*------------------------------------
        display error messages
-------------------------------------*/

display_error(char *buffer)
        {
        _setbkcolor((long)GREEN);
        _settextcolor(RED+16);
        _outtext(buffer);
        printf("\a\a\a");
        _setbkcolor((long)BLUE);
        _settextcolor(WHITE);
        _wrapon(_GWRAPON);
        _displaycursor(_GCURSOROFF);
        _outtext("\n");
        }

/*------------------------------------
        use messages area
-------------------------------------*/

use_message_window()
        {
        _settextwindow(20,3,24,78);
        _setbkcolor((long)BLUE);
        _settextcolor(WHITE);
        _wrapon(_GWRAPON);
        _displaycursor(_GCURSOROFF);
        }

/*------------------------------------
```

```
        use group display area
-------------------------------------*/

use_group_window()
        {
        _settextwindow(4,3,4,39);
        tab(3,4);
        _settextcolor(WHITE);
        _setbkcolor((long)BLUE);
        _clearscreen(_GWINDOW);
        }

/*-------------------------------------
        use time display area
-------------------------------------*/

use_time_window()
        {
        _settextwindow(4,58,4,79);
        tab(58,4);
        _settextcolor(WHITE);
        _setbkcolor((long)BLUE);
        }

/*-------------------------------------
        use status display area
-------------------------------------*/

use_status_window()
        {
        _settextwindow(18,3,18,78);
        tab(3,18);
        _settextcolor(WHITE);
        _setbkcolor((long)BLUE);
        _clearscreen(_GWINDOW);
        }

/*-------------------------------------
        use main display area
-------------------------------------*/

use_display_window()
        {
        _settextwindow(6,3,17,78);
        _settextcolor(WHITE);
        _setbkcolor((long)BLUE);
        tab(3,6);
        }

/*-------------------------------------
        move the cursor to the position (x,y)
-------------------------------------*/

tab(short x,short y)
        {
        _settextposition(y,x);
        }
```

```
/*********************************************************************

NAME            SPIRULIN.C

AUTHOR          BINOIS C       (modified by FULGET N. ADERSA)

DESCRIPTION
        MAIN PROGRAM listing file

UPDATES
        20-09-95

*********************************************************************/

#ifndef ADERSA
#include <graph.h>
#include <process.h>
#endif
#include <stdio.h>
#include <signal.h>
#include <stdlib.h>

#include "userdef.h"
#include "melissa.h"

int     my_interrupt();
char    buffer[100];
GPS_FILE *gps, *open_file_gps(), *activ_grp_gps();
int pointer;
VARS    essai;

main()
        {
#ifndef ADERSA
        void wait_time(int);
#else
        void wait_time();
#endif
        char    chr;
        int flag;

/*---------------------------------------
        screen initialisation
---------------------------------------*/

#ifndef ADERSA
        screen_init();
#endif

/*---------------------------------------
        set interruption
---------------------------------------*/

        if( signal(SIGINT,my_interrupt) == (int(*)())-1)
                {
                _outtext("\nCouldn't set SIGINT *** Program Terminated ***\n");
                exit(0);
                }


/*---------------------------------------
        timebase synchronisation
---------------------------------------*/

        wait_time(1);

/*---------------------------------------
        open groups and active one
---------------------------------------*/

        gps=open_file_gps();
        gps=activ_grp_gps();
        set_gps(gps);

/*---------------------------------------
        variables initialisation
---------------------------------------*/

        init_vars();
        init_alarm();

/*---------------------------------------
        waiting loop
---------------------------------------*/

        do
                {
                if(!timebase())
```

```
                        {
                        control_spiru();
                        flag=0;
                        }
                else
                        {
                        if(!flag)
                                {
                                alarm_spiru();
                                check_network();
                                flag=1;
                                }
                        }
                if(kbhit())
                        {
                        chr=getch();
                        }
                }
        while(1);
        }

/*------------------------------------
        interruption of programm
------------------------------------*/

int     my_interrupt()
        {
        char    ch;
        signal(SIGINT,SIG_IGN);
        _outtext("Terminate processing ? ");
        ch=getch();
        if((ch=='y')||(ch=='Y'))
                {
                close_grp_gps();
                _outtext("\nbye.... *** Program Terminated by user ***\n");
                wait_time(5);
#ifndef ADERSA
                _setvideomode(_DEFAULTMODE);
#endif
                exit(0);
                }
        if( signal(SIGINT,my_interrupt) == (int(*)())-1)
                {
                _outtext("\nCouldn't set SIGINT *** Program Terminated ***\n");
                exit(0);
                }
        _outtext("Continue...\n");
        return;
        }
```

```
/********************************************************************

        NAME            VARS.C

        AUTHOR          BINOIS C     (modified by FULGET N. ADERSA)

        DESCRIPTION
                access to data via gps functions
                and variables management

        UPDATES
                20-09-95

********************************************************************/


#ifndef ADERSA
#include <process.h>
#endif
#include <time.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <malloc.h>

#include "melissa.h"
#include "userdef.h"

static GPS_FILE *gps;
GPS_FILE *activ_grp_gps();
int my_interrupt();

/*-------------------------------------
        acquisition of gps pointer
-------------------------------------*/
#ifndef ADERSA
void set_gps(GPS_FILE *gp)
#else
void set_gps(gp)
GPS_FILE *gp ;
#endif

        {
        gps=gp;
        }


/*-------------------------------------
        read function
-------------------------------------*/
#ifndef ADERSA
void read_var(VARS *read_var)
#else
void read_var(read_var)
VARS *read_var;
#endif


        {
        double  value;
#ifndef ADERSA
        double  read_gps(VARS *);
#else
        double  read_gps();
#endif
        int     file_rank;
        char    buffer[80];

        value=read_gps(read_var);
        if(value==-1)
                {
                file_rank=gps->rank;
                while(value==-1)
                        {
                        gps=activ_grp_gps();
                        value=read_gps(read_var);
                        if(file_rank==gps->rank)
                                {
                                sprintf(buffer,"Can not find %s in gps files",read_var->name);
                                display_error(buffer);
                                my_interrupt();
                                }
                        }
                }
        read_var->i++;
        read_var->i&=NB_SAMP;
        read_var->val[read_var->i]=value;
        read_var->value=value;
        }

/*-------------------------------------
        read gps sub_function
-------------------------------------*/

#ifndef ADERSA
double read_gps(VARS *read_vars)
#else
double read_gps(read_vars)
VARS *read_vars;
#endif

        {
        OGPS    read_ogps;
        IGPS    read_igps;
        int     read_count,j,k;
        int     ret_code,tag_found;
        char    buffer[9],name[9];
        double  read_value;

        read_value=0;
        for(read_count=0;read_count<1;read_count++)
        {
        switch(read_vars->tag_cmd)
                {
                case TAG:
                        {
                        ret_code=rd_gps(read_vars->name,&read_igps);
                        if(ret_code==-1)
                                {
                                error_gps();
                                return(-1);
                                }
                        switch(ret_code)
                                {
                                case ISBIT:
                                        {
                                        if(read_igps.i.d.val_state==ACTIV)
                                                {read_value=1;}
                                        else
                                                {read_value=0;}
                                        if(!read_vars->update)
                                                {
                                                if(read_igps.i.d.act_state==ACTIV)
                                                        {read_vars->max=1;}
                                                else
                                                        {read_vars->max=0;}
                                                read_vars->dev_num=read_igps.h.dev_num;
                                                sprintf(read_vars->file,"%s",gps->file);
                                                read_vars->update=ON;
                                                }
                                        break;
                                        }
                                case ISABUS:
                                        {
                                        read_value=read_igps.i.bus.val;
                                        if(!read_vars->update)
                                                {
                                                read_vars->min=read_igps.i.bus.l;
                                                read_vars->max=read_igps.i.bus.h;
                                                sprintf(read_vars->unit,"%s",read_igps.i.bus.unit);
                                                read_vars->dev_num=read_igps.h.dev_num;
                                                sprintf(read_vars->file,"%s",gps->file);
                                                read_vars->update=ON;
                                                }
                                        break;
                                        }
                                }
                        break;
                        }
                case CMD:
                        {
                        ret_code=set_cmd(read_vars->name,&read_ogps);
                        if(ret_code==-1)
                                {
                                error_gps();
                                return(-1);
                                }
                        switch(ret_code)
                                {
                                case OSBIT:
                                case OSREGIST:
                                        {
                                        sprintf(buffer,"%s is not a valid tag name for MICON ..."
                                        ,read_vars->name);
                                        display_error(buffer);
                                        break;
                                        }
                                case OSMILOOP:
                                        {
                                        read_value=read_ogps.o.ml.val;
                                        read_vars->out=read_ogps.o.ml.out;
                                        read_vars->sp=read_ogps.o.ml.sp;
                                        if(!read_vars->update)
                                                {
                                                read_vars->min=read_ogps.o.ml.spmin;
                                                read_vars->max=read_ogps.o.ml.spmax;
                                                sprintf(read_vars->unit,"%s",read_ogps.o.ml.spunit);
                                                read_vars->dev_num=read_ogps.h.dev_num;
                                                sprintf(read_vars->file,"%s",gps->file);
                                                read_vars->update=ON;
                                                }
                                        break;
                                        }
                                case OSMILOC:
                                        {
                                        read_value=read_ogps.o.loc.val;
                                        read_vars->sp=read_ogps.o.loc.val;
                                        if(!read_vars->update)
                                                {
                                                read_vars->min=read_ogps.o.loc.locmin;
                                                read_vars->max=read_ogps.o.loc.locmax;
                                                sprintf(read_vars->unit,"%s",read_ogps.o.loc.locunit);
                                                read_vars->dev_num=read_ogps.h.dev_num;
                                                sprintf(read_vars->file,"%s",gps->file);
                                                read_vars->update=ON;
                                                }
                                        break;
                                        }
                                case OSMIVD:
                                        {
                                        read_value=read_ogps.o.vd.val;
                                        read_vars->sp=read_ogps.o.vd.val;
                                        if(!read_vars->update)
                                                {
                                                read_vars->dev_num=read_ogps.h.dev_num;
                                                sprintf(read_vars->file,"%s",gps->file);
                                                read_vars->update=ON;
                                                }
                                        break;
                                        }
                                }
                        break;
                        }
                default:
                        {
                        tag_found=OFF;
                        j=nbtags+nbcommands;
                        for(k=0;k<j;k++)
                                {
                                sprintf(buffer,"%s",gettags(k));
                                if(strcmp(buffer,read_vars->name)==0)
                                        {
                                        tag_found=ON;
                                        if(k<nbtags)
                                                {
                                                ret_code=rd_gps(read_vars->name,&read_igps);
                                                read_vars->tag_cmd=TAG;
                                                }
                                        else
                                                {
                                                ret_code=set_cmd(read_vars->name,&read_ogps);
                                                read_vars->tag_cmd=CMD;
                                                }
                                        if(ret_code==-1)
                                                {
                                                error_gps();
                                                return(-1);
                                                }
                                        else
                                                {
                                                read_vars->type=ret_code;
                                                read_count--;
                                                }
                                        break;
                                        }
                                }
                        if(!tag_found)
                                {
                                return(-1);
                                }
                        }
                }
        }
        if(read_value==-1)
```

```
                {
                return(-0.9999999);
                }
        return(read_value);
        }

/*-------------------------------------
        write function
-------------------------------------*/


#ifndef ADERSA
void write_var(VARS *write_var)
#else
write_var(write_var)
VARS *write_var;
#endif

        {
        int     file_rank,ret_code;
        char    buffer[80];

        ret_code=write_gps(write_var);
        if(ret_code==-1)
                {
                file_rank=gps->rank;
                while(ret_code==-1)
                        {
                        gps=activ_grp_gps();
                        ret_code=write_gps(write_var);
                        if(file_rank==gps->rank)
                                {
                                sprintf(buffer,"Can not find %s in gps files",write_var->name);
                                display_error(buffer);
                                my_interrupt();
                                break;
                                }
                        }
                }
        }



/*-------------------------------------
        write gps sub_function
-------------------------------------*/

#ifndef ADERSA
int write_gps(VARS *write_vars)
#else
int write_gps(write_vars)
VARS *write_vars;
#endif

        {
        OGPS    write_ogps;
        S_USER  write_s_user;
        int     ret_code,tag_written;
        time_t  ltime;
        char    buffer[80];

        tag_written=OFF;
        while(!tag_written)
        {
        switch(write_vars->tag_cmd)
                {
                case TAG:
                        {
                        sprintf(buffer,"Can't write the TAG %s ...",write_vars->name);
                        display_error(buffer);
                        my_interrupt();
                        break;
                        }
                case CMD:
                        {
                        ret_code=set_cmd(write_vars->name,&write_ogps);
                        if(ret_code==-1)
                                {
                                error_gps();
                                return(-1);
                                }
                        switch(ret_code)
                                {
                                case OSBIT:
                                case OSREGIST:
                                        {
                                        sprintf(buffer,"%s is not a valid tag name for MICON ..."
                                        ,write_vars->name);
                                        display_error(buffer);
                                        break;
                                        }
                                case OSMILOOP:
                                        {
                                        if((write_ogps.o.ml.state&16)==16)
                                                {
                                                write_s_user.mic.mode=MICLOCREM;
                                                write_s_user.mic.status_sta=ACTIV;
                                                if(wr_gps(&write_ogps,&write_s_user)==-1)
                                                        {
                                                        error_gps();
                                                        }
                                                break;
                                                }
                                        if((write_ogps.o.ml.state&1)==0)
                                                {
                                                write_s_user.mic.mode=MICAUTO;
                                                write_s_user.mic.status_sta=ACTIV;
                                                if(wr_gps(&write_ogps,&write_s_user)==-1)
                                                        {
                                                        error_gps();
                                                        }
                                                break;
                                                }
                                        write_s_user.mic.status_sta=INACTIV;
                                        write_s_user.mic.status_out=INACTIV;
                                        write_s_user.mic.status_ra=INACTIV;
                                        write_s_user.mic.status_bi=INACTIV;
                                        write_s_user.mic.status_loc=INACTIV;
                                        write_s_user.mic.status_vd=INACTIV;
                                        write_s_user.mic.val_sp=write_vars->sp;
                                        write_s_user.mic.status_sp=ACTIV;
                                        tag_written=ON;
                                        break;
                                        }
                                case OSMILOC:
                                        {
                                        write_s_user.mic.val_loc=write_vars->sp;
                                        write_s_user.mic.status_loc=ACTIV;
                                        tag_written=ON;
                                        break;
                                        }
                                case OSMIVD:
                                        {
                                        write_s_user.mic.val_vd=(int)write_vars->sp;
                                        write_s_user.mic.status_vd=ACTIV;
                                        tag_written=ON;
                                        break;
                                        }
                                }
                        break;
                        }
                default:
                        {
                        read_var(write_vars);
                        }
                }
        }
        if(wr_gps(&write_ogps,&write_s_user)==-1)
                {
                error_gps();
                time(&ltime);
                sprintf(buffer,"When writing %s at time %s",write_vars->name,
                ctime(&ltime));
                display_error(buffer);
                return(0);
                }
        else
                {
                return(0);
                }
        }

/*---------------------------------------
        errors management
---------------------------------------*/

error_gps()
        {
        switch (gpserror)
                {
                case ERDOS:
                        {
                        display_error("A DOS problem has occured ...\n");
                        my_interrupt();
                        break;
                        }
                case ENETW:
                        {
                        display_error("Network or Mailbox fault ...\n");
                        break;
                        }
                case INVALID_FGPS:
                        {
                        display_error("Incorrect GPS file ...\n");
                        my_interrupt();
                        break;
                        }
                case ETAGCMD:
                        {
                        break;
                        }
                case ETAGTYP:
                case ECMDTYP:
                        {
                        display_error("Unknown CMD or TAG ...");
                        my_interrupt();
                        break;
                        }
                case ECONV:
                        {
                        display_error("Floating point conversion error:ignored ...");
                    /* my_interrupt();*/
                        break;
                        }
                case EEQUIP:
                        {
                        display_error("Unknown PLC protocol ...");
                        my_interrupt();
                        break;
                        }
                case ENUMPLC:
                        {
                        display_error("Invalid device number ...");
                        my_interrupt();
                        break;
                        }
                default:
                        {
                        display_error("Unidentified error ...");
                        my_interrupt();
                        break;
                        }
                }
        }

/*-------------------------------------
        check if mailbox is refresh
-------------------------------------*/

check_network()
        {
        display_status("Checking Network ...");
        while(garde(101,1))
                {
                }
        display_status(" ");
        }
```

```
/*********************************************************************

        NAME            MELISSA.H

        AUTHOR          BINOIS C   (modified by FULGET N. ADERSA)

        DESCRIPTION
                General Declarations

        UPDATES
                20-09-95

*********************************************************************/

/*---------------------------------------
        constants for VARS
---------------------------------------*/
#define TAG     128
#define CMD     255
#define UNDEF   0
#define NB_SAMP 0xFF            /* number of samples stored in val[ ] */


/*---------------------------------------
        structure for variables
---------------------------------------*/

typedef struct _vars {
        char    name[9];        /* tag name             */
        char    file[12];       /* file name            */
        int     type;           /* rd_gps/set_cmd return code           */
        unsigned char tag_cmd;  /* TAG or CMD           */
        unsigned int dev_num;   /* controller number    */
        double  value;          /* current value        */
        int     i;              /* pointer on last value entered in val[] */
        double  val[NB_SAMP+1]; /* previous values      */
        double  min;
        double  max;
        double  sp;             /* set point for LOOP       */
        double  out;            /* out value for LOOP       */
        char    unit[5];        /* unit for analog values   */
        char    update;         /* ON when structure updated*/
} VARS;

/*---------------------------------------
        structure for opened GPS files
---------------------------------------*/

typedef struct _gps_file{
        char    file[15];           /* file name                */
        int     handler;            /* handler of gps file      */
        int     rank;               /* rank of the gps file     */
        struct  _gps_file   *next;  /* next opened gps file     */
} GPS_FILE;


/*---------------------------------------
        structure for reactor state
---------------------------------------*/

typedef struct  _react{
        double Cxa;     /* in mg/l      */
        double Cno3;    /* in mg/l      */
        double temp;    /* in degre C   */
        double press;   /* in           */
        double Eb;      /* in W/m2      */
        double Fr;      /* in W/m2      */
        double rxa;     /* in mg/l/h    */
        double rn;      /* in mg/l/h    */
        double ro2;     /* in mg/l/h    */
} REACT;

/*---------------------------------------
        general constants
---------------------------------------*/

#define TSAMP   60              /* sampling interval in secondes        */
#define SYNCHRO  1
#define ERROR_SPEED 0.01        /* max error for the model      */
#define VOLUME_LIGHT 3.900      /* illuminated volume (in l)    */
#define VOLUME_TOTAL 7.000      /* total volume (in l)          */

/*---------------------------------------
        Model control constants
---------------------------------------*/

#define USE_FR  10
#define USE_EB  20
```

```
/*-------------------------------------
        Mathematical constants
-------------------------------------*/

#define PI      3.14159265359


/*-------------------------------------
        colours
-------------------------------------*/

#define BLACK   0
#define BLUE    1
#define GREEN   2
#define CYAN    3
#define RED     4
#define MAGENTA 5
#define BROWN   6
#define WHITE   7

/*-------------------------------------
        ADERSA constants
-------------------------------------*/

#define DT      30      /* sampling period of PFC        */
#define NHC     5       /* coincidence point in DT       */
#define LAMBDA  0.75    /* reference trajectory dynamic */
#define DFR     10.     /* radiant flux increment W/m2  */
#define FR_MIN  10.     /* min constraint on Fr W/m2    */
#define FR_MAX  400.    /* max constraint on Fr W/m2    */
#define DQ      0.1     /* flow variation                */
#define CXA_MIN 250.    /* min constaint on cxa mg/l    */
#define CXA_MAX 1500.   /* max constaint on cxa mg/l    */
```

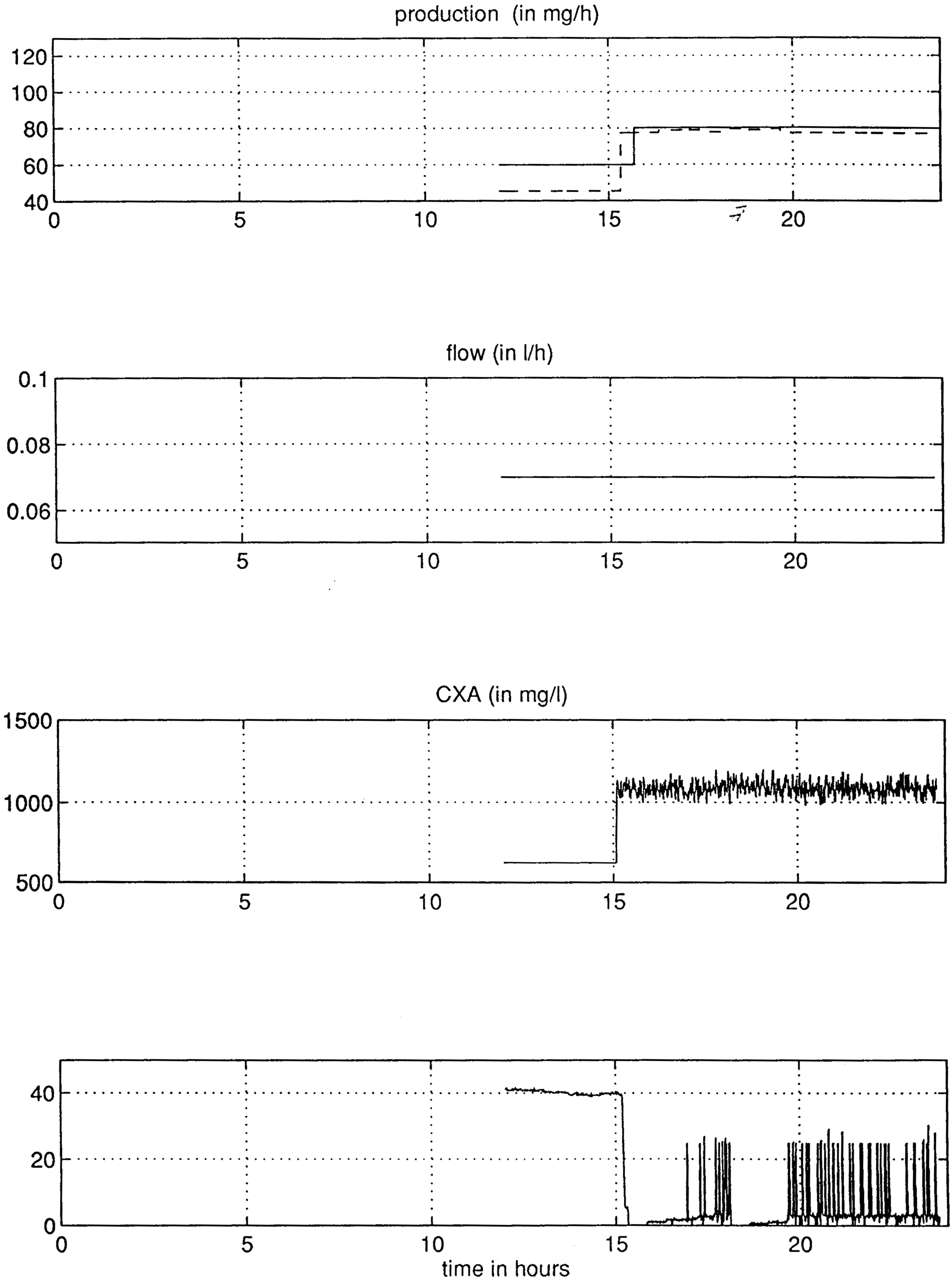
```
/************************************************************************

NAME            USERDEF.H

AUTHORS         (C) TOPTOOLS 1988,1989,1990

DESCRIPTION

        INDUSTAR General Purpose Station - User Include File

UPDATES
        90-05-10 - Add TiWay support

************************************************************************/


/* ---------------------------
        Miscellaneous constants definition
--------------------------- */

#define ACTIV            1                      /* command is active                                   */
#define INACTIV          0                      /* command is inactive                                 */

#define ON                       1                      /* ON  value  for digital commands             */
#define OFF                      0                      /* OFF value  "      "      "                  */


/* ---------------------------
        rd_gps() return codes
--------------------------- */

                                                                /* Tag type                                      Structure
/
#define ISBIT            1                      /* digital                                             IDIG    */
#define ISABUS           2                      /* analog                                              IBUSA   */


/* ---------------------------
        set_cmd() return codes
--------------------------- */

                                                                /* Cmd type                                      Structure
/
#define OSBIT            32                     /* digital                                             ODIG    */
#define OSMILOOP         41                     /* analog  (Micon loop)                OMLO    */
#define OSREGUL          42                     /* analog  (AB,TCS,PLS loop)   OREGU   */
#define OSREGIST         52                     /* analog  (register)                  OREGIS  */
#define OSMIVD           61                     /* digital (Micon DV)                  OMVD    */
#define OSMILOC          71                     /* analog  (Micon loc)                 OMLOC   */


/* ---------------------------
        gpserror values
--------------------------- */

#define ERDOS            1                      /* DOS problem                                                 */
#define ENETW            2                      /* network or mailbox                                  */
#define INVALID_FGPS 3                  /* invalid GPS file                                    */
#define ETAGCMD          4                      /* tag or command not found                            */
#define EVARCOD          5                      /* invalid variable codification               */
#define ECONV            6                      /* floating point conversion                   */
#define ETAGTYP          7                      /* unknown tag type                                            */
#define EEQUIP           8                      /* unknown PLC protocol                                */
#define ECMDTYP          9                      /* unknown command type                                */
#define ENUMPLC          10                     /* invalid device number                               */
#define ERPROTECT        11                     /* protection key not found                            */
#define ENSECTOR         12                     /* invalid record number                               */
#define ETYPVARCAL       13                     /* not a computed variable                             */
#define EGDPREV          24                     /* action on previous GD var not over  */
#define EGDTYPVAR        25                     /* not a GD variable                                   */
#define EGDPOLLIS        26                     /* POLLIS.TAB not found                                */
#define EGDCMDFAIL       27                     /* failed command to GD                                */
#define ELOCAL           31                     /* PLC in Local state (cmd impossible) */
#define ISMODAUTO        32                     /* AUTO mode : change is impossible            */
#define EPLSOVER         44                     /* PLStar variable value out of limits */


/* ---------------------------
        Not selected command fields
--------------------------- */

#define FVALNOTUSED      0xF4240        /* 4bytes, not selected analog cmd field*/
#define IVALNOTUSED 64                  /* 1byte,  not selected dig or loop fld.*/


/* ---------------------------
        Specific definitions for Micon equipments
--------------------------- */
```

```
#define MICLOCREM       151                     /* local/remote                                                 */
#define MICCASCA        150                     /* cascade                                                              */
#define MICAUTO         149                     /* auto                                                                 */
#define MICMANUAL       148                     /* manual                                                               */
#define ISINCONFIG      155                     /* Micon is starting configuration              */


/* ---------------------------
        External declarations
--------------------------- */

extern int gpserror;                    /* variable to house error codes                */
extern int echonetw ;                   /* network error code, when applicable  */
extern unsigned int nbtags;             /* # tags in activated group                    */
extern unsigned int nbcommands; /* # cmds in activated group                    */
extern char *gettags () ;               /* tag or command name (8-char string)  */


/* -----------------------------------------------------------------------

                                                  STRUCTURES FOR READING TAGS

----------------------------------------------------------------------- */


/* ---------------------------
        Common header structure (tags and commands)
--------------------------- */

typedef struct _iohead  {

        char      tag[9];                       /* tag or cmd                                                   */
        char      tag_name[31];         /*  "       " name                                              */
        unsigned  char tag_type;        /* tag or cmd type :
                                                                        = 1  digital input
                                                                        = 2  analog    "
                                                                        = 3  digital output
                                                                        = 4  analog      "              (loop)
                                                                        = 5    "         "              (register)
                                                                        = 6    "         "              Micon  VD
                                                                        = 7    "         "              Micon  LOC
                                                                        ...................................     */

        unsigned char   zone;           /* INDUSTAR C&C Station number                  */
        unsigned int    dev_num;        /* controller number                                    */
        unsigned char   dev_type;       /* controller type :
                                                                        = 1  MICON
                                                                        = 2  JBUS-MODBUS
                                                                        = 3  STRUTHERS & DUNN
                                                                        = 4  ALLEN BRADLEY
                                                                        = 5  UNITELWAY
                                                                        = 8  TCS 6000
                                                                        = 9  TIWAY
                                                                        = 13 LAC
                                                                        = 14 PLStar TT
                                                                        = 15 Ghost Device TT
                                                                        ...................................     */

        unsigned char   log_can;        /*   logical channel number                             */

} IOHEAD;


/* ---------------------------
        Digital Input
--------------------------- */

typedef struct _idig {

        unsigned char val_state;                /* tag state  (0/1)                                     */
        unsigned char act_state;                /* active state                                         */
        char              alarm_tag[9];         /* associated alarm tag                         */
        char              fault_tag[9];         /* associated fault tag                         */
        char              progress_tag[9];      /* associated in progress tag           */

        char              var_nam[7] ;          /* TIWAY - process variable name        */
        int                       var_typ ;                     /* TIWAY - process variable type        */
        int                       var_num ;                     /* TIWAY - process variable #           */

} IDIG;


/* ---------------------------
        Analog tag
--------------------------- */

typedef struct _ibusa {
        float             val;                          /* value  (IEEE floating point)          */
        float             scale;                        /* scale                                                */
        float             vl;                           /* very low limit                               */
        float             l;                            /* low limit                                    */
        float             h;                            /* high limit                                   */
        float             vh;                           /* very high limit                              */
        char              unit[5];                      /* unit                                                 */

        char              var_nam[7] ;          /* TIWAY - process variable name        */
        int                       var_typ ;                     /* TIWAY - process variable type        */
        int                       var_num ;                     /* TIWAY - process variable #           */

} IBUSA;


/* ---------------------------
        Digital or Analog tag (without the header structure)
--------------------------- */

typedef union _u_inp {

        IDIG    d;                                              /* when digital                                         */
        IBUSA   bus;                                    /* when analog                                  */

} U_INP;


/* ---------------------------
        Structure for the calls to rd_gps()
--------------------------- */

typedef struct _igps {

        IOHEAD  h;                                              /* common header structure                      */
        U_INP   i;                                              /* according to tag type and PLC        */

} IGPS;


/* ----------------------------------------------------------------------

                                   STRUTURES FOR GETTING COMMAND INFORMATION

---------------------------------------------------------------------- */


/* ---------------------------
        Digital command
--------------------------- */

typedef struct _odig {

        char    cmd_tag[9];                     /* tag name associated to the command   */
        char    st_cmd;                         /* tag  state (0/1)                                     */
        char    as_cmd;                         /* active state                                         */
        char    cmd_file;                       /* file number for ALLEN BRADLEY                */
        int             cmd_typ ;                       /* TIWAY
/

        char    plcloc_tag[9];          /* local/remote tag                                     */
        char    st_plcloc;                      /* local(0)/remote(1) tag state         */
        char    as_plcloc;                      /* active state                                         */

        char    alarm_tag[9];           /* alarm tag associated to the command  */
        char    st_alarm;                       /* state alarm tag (0/1)                        */
        char    as_alarm;                       /* active state                                         */

        char    fault_tag[9];           /* fault tag                                            */
        char    st_fault;                       /* state fault tag (0/1)                        */
        char    as_fault;                       /* active state                                         */

        char    instart_tag[9];         /* in progress tag                                      */
        char    st_instart;                     /* state in progress tag (0/1)          */
        char    as_instart;                     /* active state                                         */

        char    inhalt_tag[9];          /* in halt tag                                          */
        char    st_inhalt;                      /* state in halt tag (0/1)                      */
        char    as_inhalt;                      /* active state                                         */

        char    localcmd_tag[9];        /* local command tag (element)          */
        char    st_localcmd;            /* state local command tag (0/1)        */
        char    as_localcmd;            /* active state                                         */

        char    cmdtype;                        /* command type = 0,1,2,3                       */
        char    mult_tab;                       /* not used by the G.P.S.                       */

        unsigned devon_adr;                     /* PLC bit state adresse ON                     */
        char     as_devon;                        /* active state for ON                              */
        char     mask_on;                         /* bit mask for ALLEN BRADLEY               */

        unsigned devoff_adr;              /* PLC bit state addresse OFF               */
        char     as_devoff;                       /* active state for OFF                             */
        char     mask_off;                        /* bit mask for ALLEN BRADLEY               */

} ODIG;


/* ---------------------------
        Analog command (register)
--------------------------- */

typedef struct _oregis {

        char     cmd_tag[9];              /* tag name associated to the command   */
        float    val;                             /* tag measure value                                */
        int              cmd_typ ;                        /* TIWAY
/

        unsigned char cmd_mod;            /* mode variable PLStar                             */
        char     locrem_tag[9];           /* tag for local/remote state               */
        char     state;                           /* PLC state Local(0)/Remote(1)             */
        char     as_locrem;                       /* active state Local/Remote                */

        char     reg1_tag[9];             /* tag register 1                                   */
        float    reg1_val;                        /* value register 1                                         */
        float    reg1_min;                        /* value mini  reg. 1                               */
        float    reg1_max;                        /* value maxi  reg. 1                               */
        char     reg1_unit[5];            /* unit register 1                                  */
        unsigned reg1_adrhx;              /* PLC adress (hexa) of register 1          */
        unsigned char reg1_mod;           /* mode variable PLStar                             */
        char     reg1_file;                       /* # file AB                                                */
        int              reg1_typ ;                       /* TIWAY
/
        int              reg1_num ;                       /* TIWAY
/

        char     reg2_tag[9];             /* tag register 2                                   */
        float    reg2_val;                        /* value register 2                                         */
        float    reg2_min;                        /* value mini  reg. 2                               */
        float    reg2_max;                        /* value maxi  reg. 2                               */
        char     reg2_unit[5];            /* unit reg. 2                                      */
        unsigned reg2_adrhx;              /* PLC adress (hexa) of register 2          */
        unsigned char reg2_mod;           /* mode variable PLStar                             */
        char     reg2_file;                       /* # file AB                                                */
        int              reg2_typ ;                       /* TIWAY
/
        int              reg2_num ;                       /* TIWAY
/

        char     reg3_tag[9];             /* tag register 3                                   */
        float    reg3_val;                        /* value register 3                                         */
        float    reg3_min;                        /* valeur mini reg. 3                               */
        float    reg3_max;                        /* valeur maxi reg. 3                               */
        char     reg3_unit[5];            /* unit register 3                                  */
        unsigned reg3_adrhx;              /* PLC adress (hexa) of register 3          */
        unsigned char reg3_mod;           /* mode variable PLStar                             */
        char     reg3_file;                       /* # file AB                                                */
        int              reg3_typ ;                       /* TIWAY
/
        int              reg3_num ;                       /* TIWAY
/

} OREGIS;


/* ---------------------------
        Analog command (loop)
--------------------------- */

typedef struct _oregu {

        char     cmd_tag[9];              /* tag name associated to the command   */
        float    val;                             /* tag measure value                                */
        int              cmd_typ ;                        /* TIWAY
/
        int              tiloopnb ;                       /* TIWAY
/
        char     cmd_mod;                         /* mode variable PLStar                             */

        char     spt_tag[9];              /* tag for setpoint                                 */
        float    spt_val;                         /* setpoint value                                           */
        float    spt_min;                         /* value mini  spt                                          */
        float    spt_max;                         /* value maxi  spt                                          */
        char     spt_unit[5];             /* unit for setpoint                        */
        unsigned spt_adr;                         /* adresse ALLEN B                                          */
        unsigned spt_file;                      /*  file number of spt value (A-B)              */
        char     spt_mod;                       /* mode variable PLStar (cf. PLStar)    */
        int              spt_typ ;                      /* TIWAY
/
        int              spt_num ;                      /* TIWAY
/


        char     mo_tag[9];                     /* tag for Manual Output                                */
        float    mo_val;                        /* M.O. value                                                   */
        float    mo_min;                        /* value mini  M.O.                                             */
        float    mo_max;                        /* value maxi  M.O.                                             */
        char     mo_unit[5];            /* unit  M.O.                                                   */
        unsigned mo_adr;                        /* adresse ALLEN B                                              */
        unsigned mo_file;                       /* file number of M.O.                                  */
        char     mo_mod;                        /* mode variable PLStar (cf.PLStar)             */
        int              mo_typ ;                       /* TIWAY
/
        int              mo_num ;                       /* TIWAY
/

        char     stalo_tag[9];          /* tag for loop status                                  */
        char     stalo_mod;                     /* status loop value                                    */
        char     stalo_unit[5];         /* unit for loop status                                 */
        unsigned stalo_adr;                     /* adresse ALLEN B                                              */
        unsigned stalo_file;            /* file number of loop status                   */
        char     stat_mod;                      /* mode variable PLStar (cf. PLStar)    */
        int              stalo_typ ;            /* TIWAY                                                                */
        int              stalo_num ;            /* TIWAY                                                                */

} OREGU;


/* ---------------------------
        Analog command (Micon loop)
--------------------------- */


typedef struct _omlo {

        char     regutag[9];            /* tag associated to the command                */
        float    val;                           /* current value read in the mailbox    */
        int              nloop;                         /* loop number                                                  */
        float    sp;                            /* setpoint value                                               */
        float    spmin;                         /*   "     minimum value                                */
        float    spmax;                         /*   "     maximum  "                                  */
        char     spunit[5];                     /*  unit for setpoint                                   */
        float    out;                           /* outpoint value                                               */
        float    outmin;                        /*   "     minimum value                                */
        float    outmax;                        /*   "     maximum  "                                  */
        char     outunit[5];            /* unit for outpoint                                    */
        float    ratio;                         /* ratio value                                                  */
        float    ratiomin;                      /*   "     mini                                                 */
        float    ratiomax;                      /*   "     maxi                                                 */
        char     ratiounit[5];          /*  unit for ratio                                              */
        float    bias;                          /* bias value                                                   */
        float    biasmin;                       /*   "     mini                                                 */
        float    biasmax;                       /*   "     maxi                                                 */
        char     biasunit[5];           /*   unit for bias                                              */
        unsigned char state;            /* loop state (auto/manual/cascade)             */

} OMLO;


/* ---------------------------
        Analog command (Micon LOC)
--------------------------- */

typedef struct _omloc {

        char     loctag[9];                     /* tag name associated to the command   */
        float    val;                           /* current LOC value                                    */
        int              nloc;                          /* LOC number                                                   */
        float    locmin;                        /*  minimum LOC value                                   */
        float    locmax;                        /*  maximum  "    "                                             */
        char     locunit[5];            /*  unit for LOC                                                */

} OMLOC;


/* ---------------------------
        Digital command (Micon DV)
--------------------------- */

typedef struct _omvd {

        char     vdtag[9];                      /* tag name associated to the command   */
        int              val;                           /* current Discrete Virtual value               */
        int             nvd;                            /* Discrete Virtual number                              */

} OMVD;


/* --------------------------
        Digital or Analog command (without the header)
-------------------------- */

typedef union _u_outp {

        ODIG    d;                                      /* digital (common for all PLCs)                */
        OMLO    ml;                                     /* Loop: MICON                                                  */
        OREGU   l;                                      /* Loop: ALLEN-BRADLEY, PLStar, TIWAY   */
        OREGIS  r;                                      /* Register: MODBUS, JBUS, LAC, PLStar  */
                                                                /* ...ALLEN-B, UNITELWAY, TIWAY                 */
        OMVD    vd;                                     /* MICON VD
/
        OMLOC   loc;                            /* MICON LOC                                                    */

} U_OUTP;


/* --------------------------
        Command structure for the calls to set_cmd() and wr_gps()
-------------------------- */

typedef struct _ogps {

        IOHEAD  h;                                      /* common header structure                              */
        U_OUTP  o;                                      /* according to the PLC                                 */

} OGPS;


/* ---------------------------------------------------------------------

                                        STRUCTURES TO SEND COMMANDS

--------------------------------------------------------------------- */


/* --------------------------
        Sending a command to JBUS, MODBUS, LAC equipments
-------------------------- */

typedef struct _usjbus {

        char    status_bit;                     /* cmd state switch ACTIV/INACTIV (default)     */
        char    action_bit;                     /* new bit value (ON / OFF)                             */

        char    status_reg1;            /* cmd reg.1 switch ACTIV/INACTIV (default)     */
        float   newreg1;                        /* new  register 1 value                                */

        char    status_reg2;            /* cmd reg. 2 switch ACTIV/INACTIV      */
        float   newreg2;                        /* new  register 2 value                                */

        char    status_reg3;            /* cmd reg. 3 switch ACTIV/INACTIV      */
        float   newreg3;                        /* new register 3 value                                 */

} USJBUS;


/* --------------------------
        Send a command to a Micon equipment
-------------------------- */

typedef struct _usmloop {

        char     status_sp;                     /* cmd setpoint switch ACTIV/INACTIV(default)   */
        float    val_sp;                        /* new setpoint value                                   */

        char     status_out;            /* cmd outpoint switch ACTIV/INACTIV    */
        float    val_out;                       /* new outpoint value                                   */

        char     status_ra;                     /* cmd ratio switch ACTIV/INACTIV               */
        float    val_ra;                        /* new ratio value                                              */

        char     status_bi;                     /* cmd bias switch ACTIV/INACTIV                */
        float    val_bi;                        /* new bias value                                               */

        char     status_sta;            /* cmd state switch ACTIV/INACTIV               */
        unsigned char mode;                     /* new channel state value - use above MICON
                                                                        definitions (MICLOCREM etc.)                    */

        char     status_loc;            /* cmd LOC switch ACTIV/INACTIV                 */
        float    val_loc;                       /* new LOC value                                                */
        char     status_vd;                        /* cmd DV switch ACTIV/INACTIV                   */
        char     val_vd;                           /* new DV value  ON/OFF                                  */

} USMLOOP;


/* ----------------------------
        Send a command to :
        Allen-Bradley, PLStar, UnitelWay, Reliance, TCS or TIWAY equipment
---------------------------- */

typedef struct _usalb {

        char     status_sp1;               /* cmd setpoint or register 1 switch ACTIV/INACTIV(default)      */
        float    val_sp1;                          /* new setpoint or register 1 value              */

        char     status_out2;              /* cmd outpoint or register 2 switch ACTIV/INACTIV       */
        float    val_out2;                         /* new outpoint or register 2 value              */

        char     status_sta3;              /* cmd loop state or register 3 switch ACTIV/INACTIV     */
        float    val_sta3;                         /* new loop state value or register 3    */

        char     status_bit;               /* cmd state switch ACTIV/INACTIV                */
        char     val_bit;                          /* new bit value  ON/OFF                                 */

} USALB;


/* ----------------------------
        Send a command to the Mailbox
---------------------------- */

typedef struct _uscalc {

        char     status_bit;               /* state switch (ACTIV/INACTIV) for new bit value        */
        char     val_bit;                          /* new bit value                                               */

        char     status_num;               /* state switch (ACTIV/INACTIV) for new analog value     */
        float    val_num;                          /* new analog value                                            */

        char     destable[32];             /* list of zones to send the message    */

} USCALC;


/* ----------------------------
        Structure for the calls to wr_gps()
---------------------------- */

typedef union _s_user {

        USJBUS   jb;               /* MODBUS,JBUS,LAC                                                                */
        USMLOOP  mic;      /* MICON                                                                                  */
        USALB    ab;               /* ALLEN BRADLEY,PLStar,UNITELWAY,TCS,RELIANCE,TIWAY*/
        USCALC   cal;      /* Ghost Device TT                                                                */

} S_USER;


/* ----------------------------
        GPS functions declarations
---------------------------- */

/* extern  int open_gr(char *name_gr); */
/* extern  int activ_gr(int h_gr); */
/* extern  int rd_gps(char *mtag,struct _igps *_igps); */
/* extern  int set_cmd(char *nom,struct _ogps *ogps); */
/* extern  int wr_gps(struct _ogps *stout,union _s_user *stuser); */
/* extern  int close_gr(int h_gr); */
/* extern  char *gettags(unsigned int ii); */
/* extern  int garde(unsigned int n,unsigned int nsecs); */
```

Annex E :

Experimental results

7



production (in mg/h)

flow (in l/h)

CXA (in mg/l)

figure E1: experimental results of 08/03

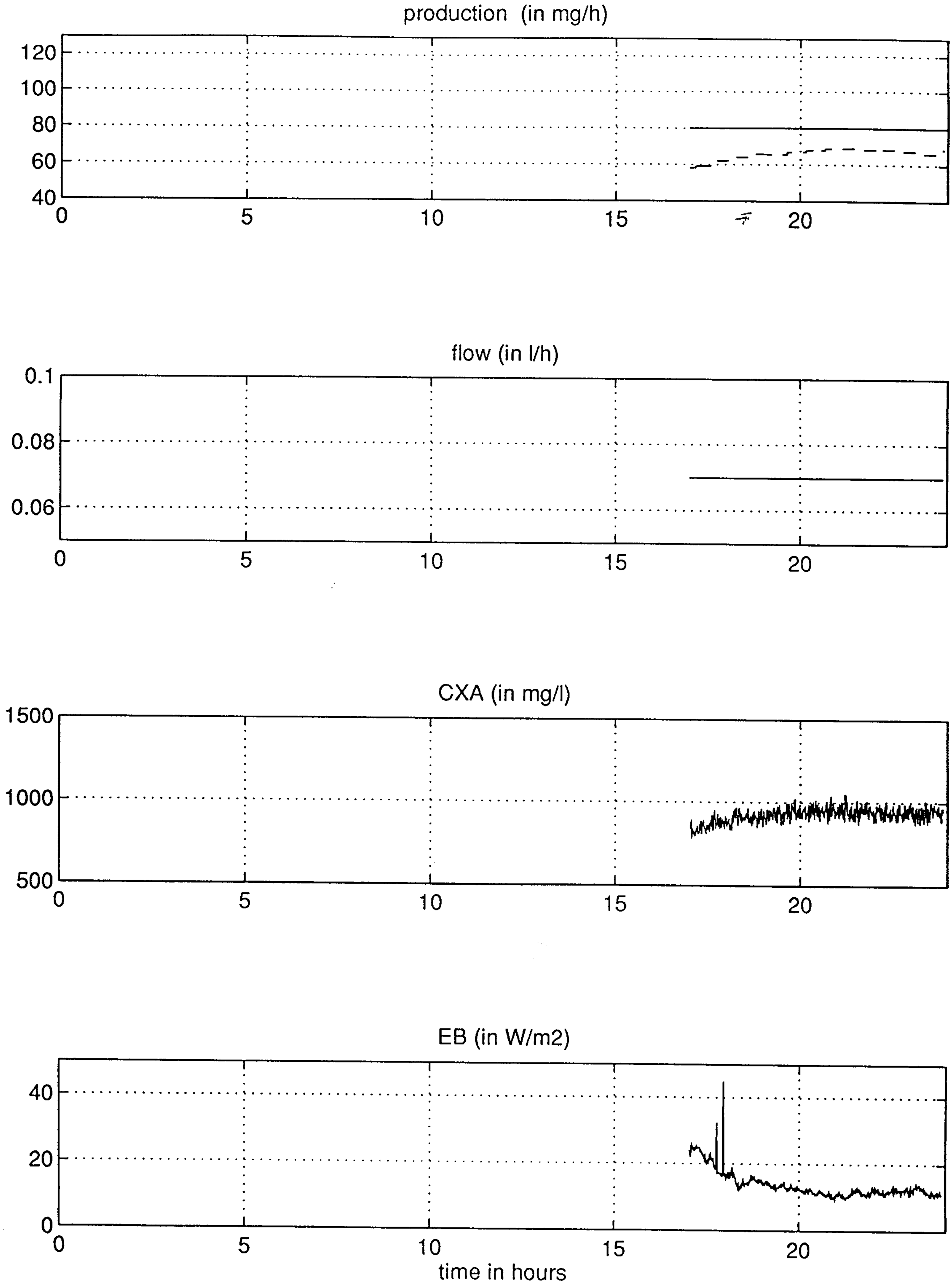


figure E2: experimental results of 09/03

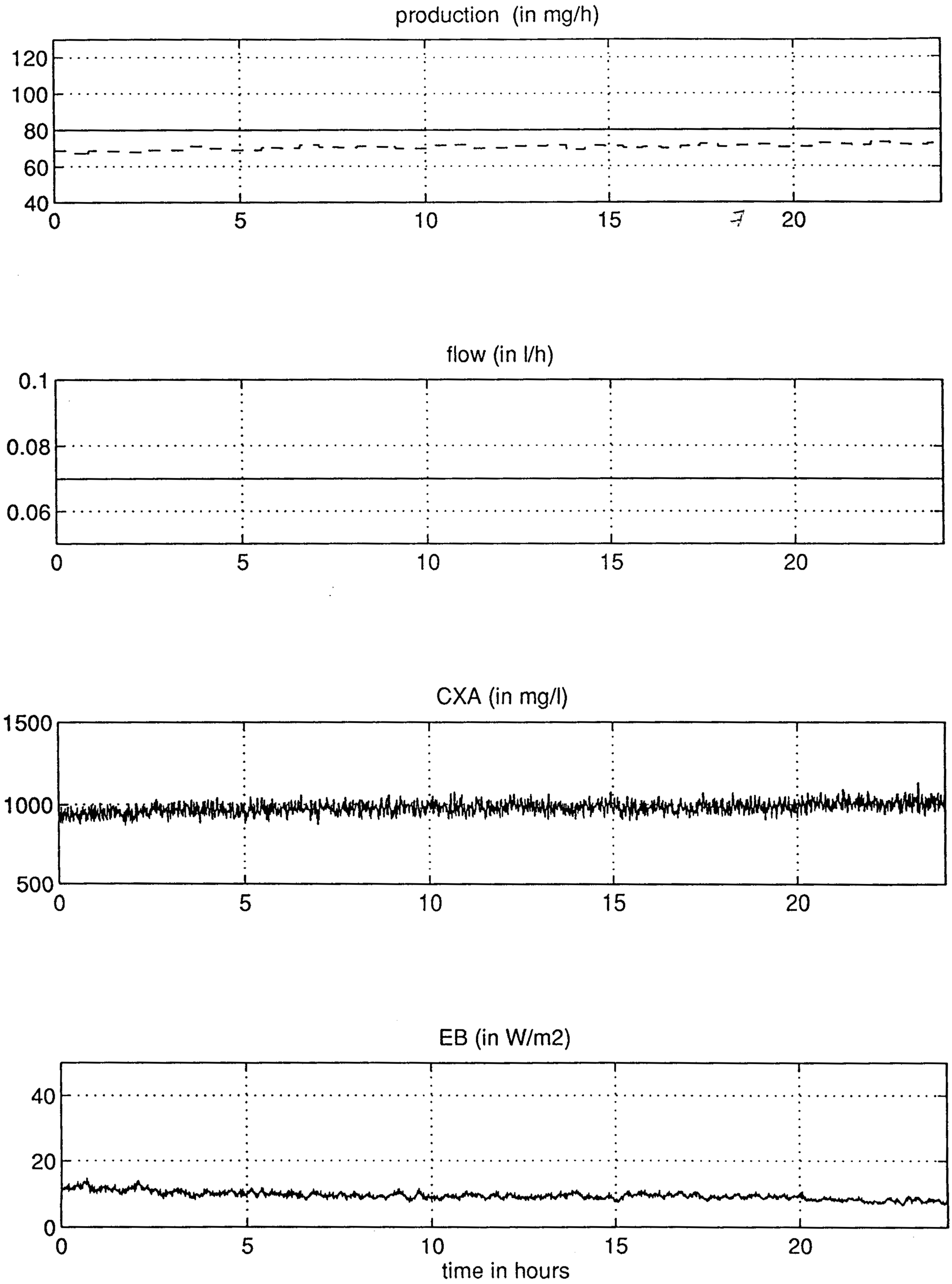


figure E3: experimental results of 10/03

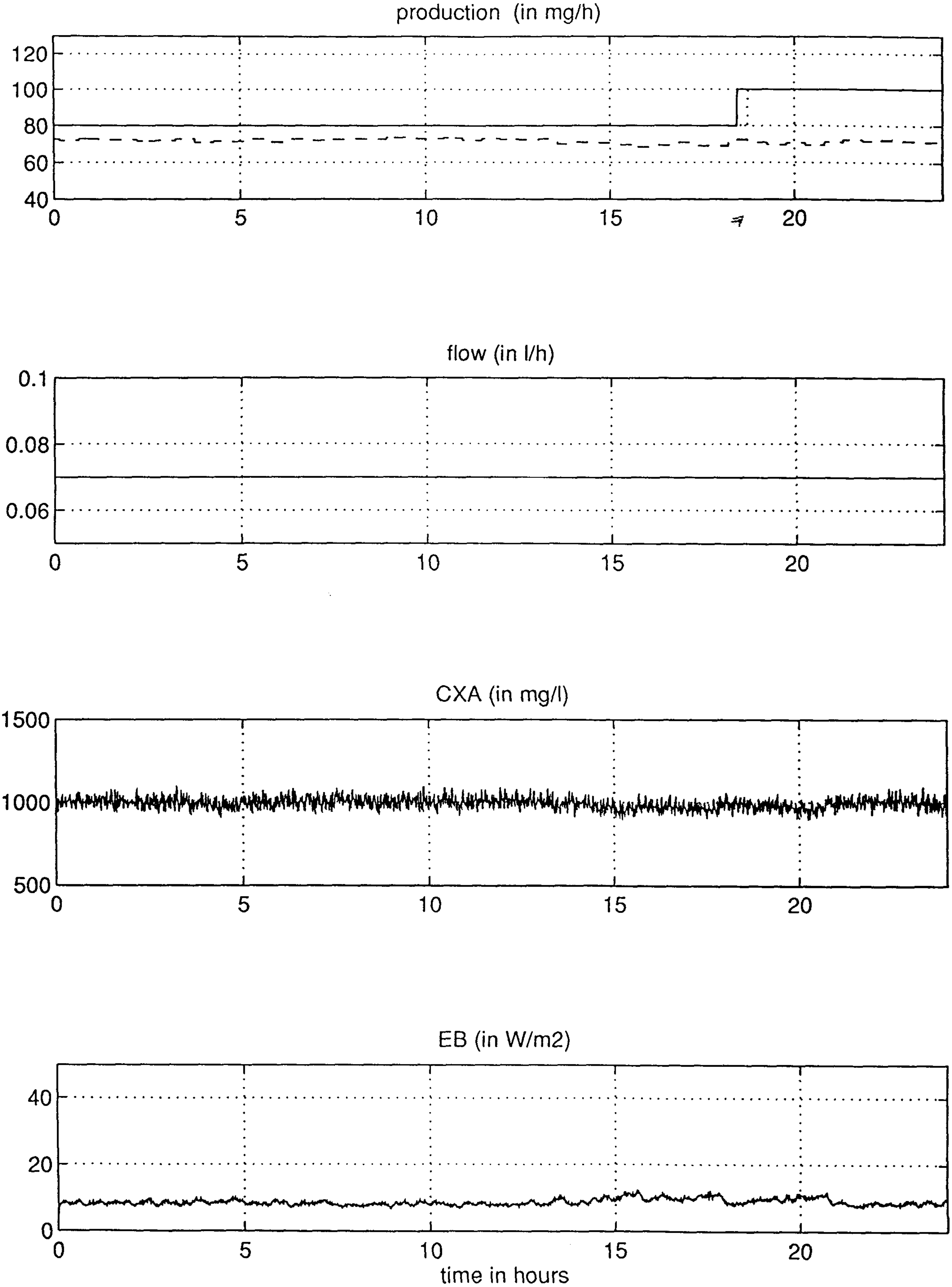


figure E4: experimental results of 11/03

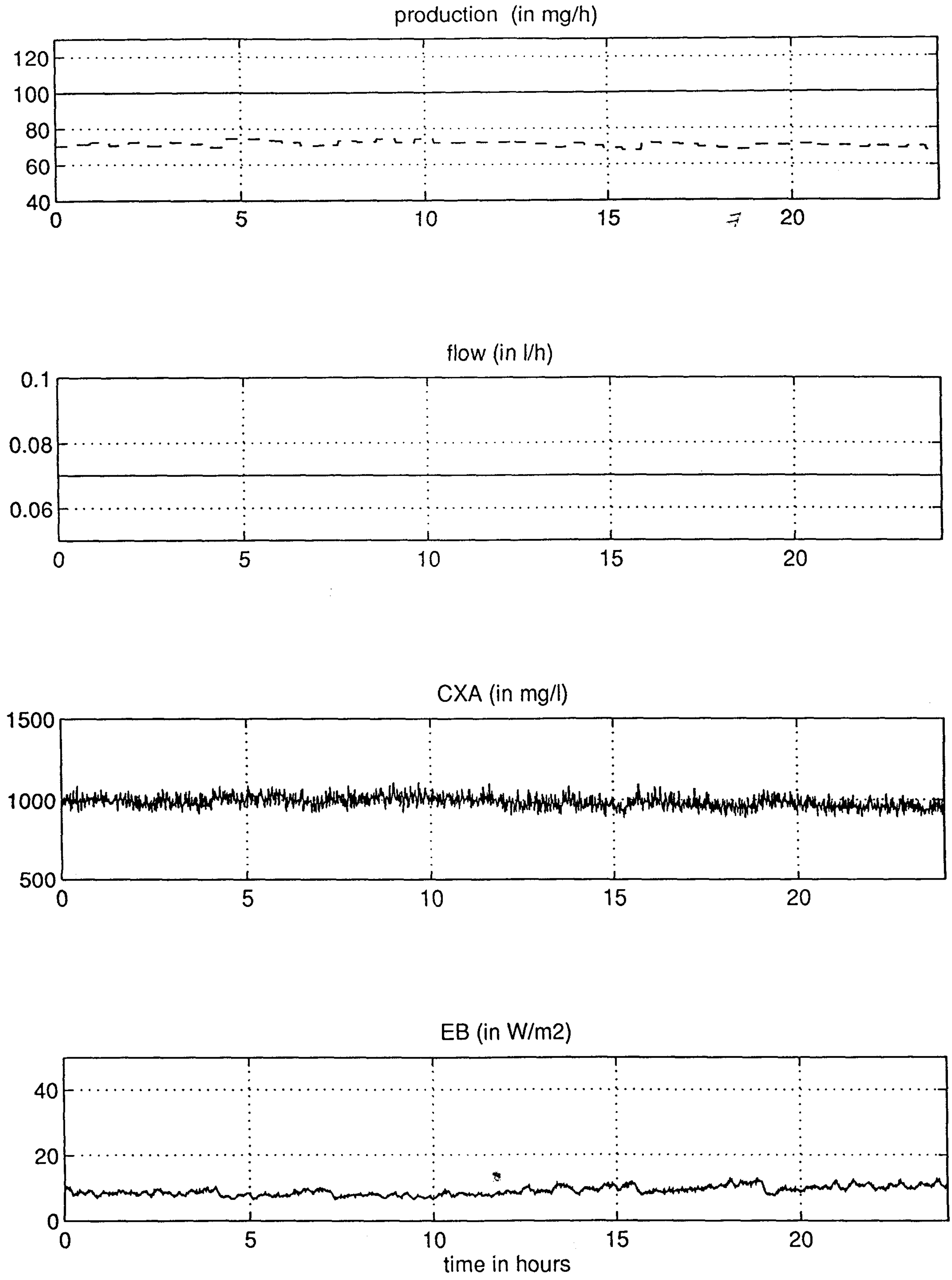
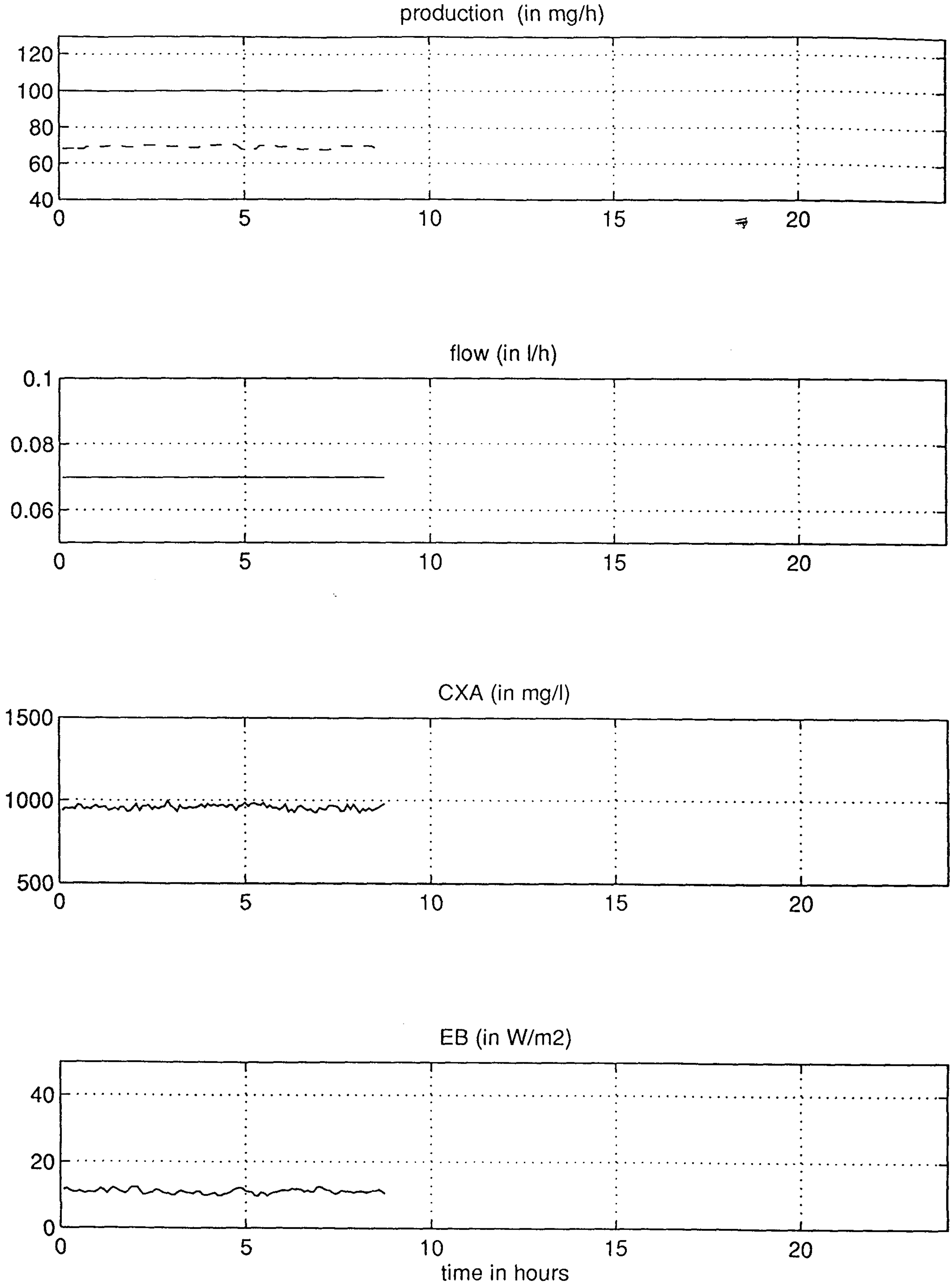


figure E5: experimental results of 12/03



production (in mg/h)

flow (in l/h)

CXA (in mg/l)

EB (in W/m2)

figure E6: experimental results of 13/03

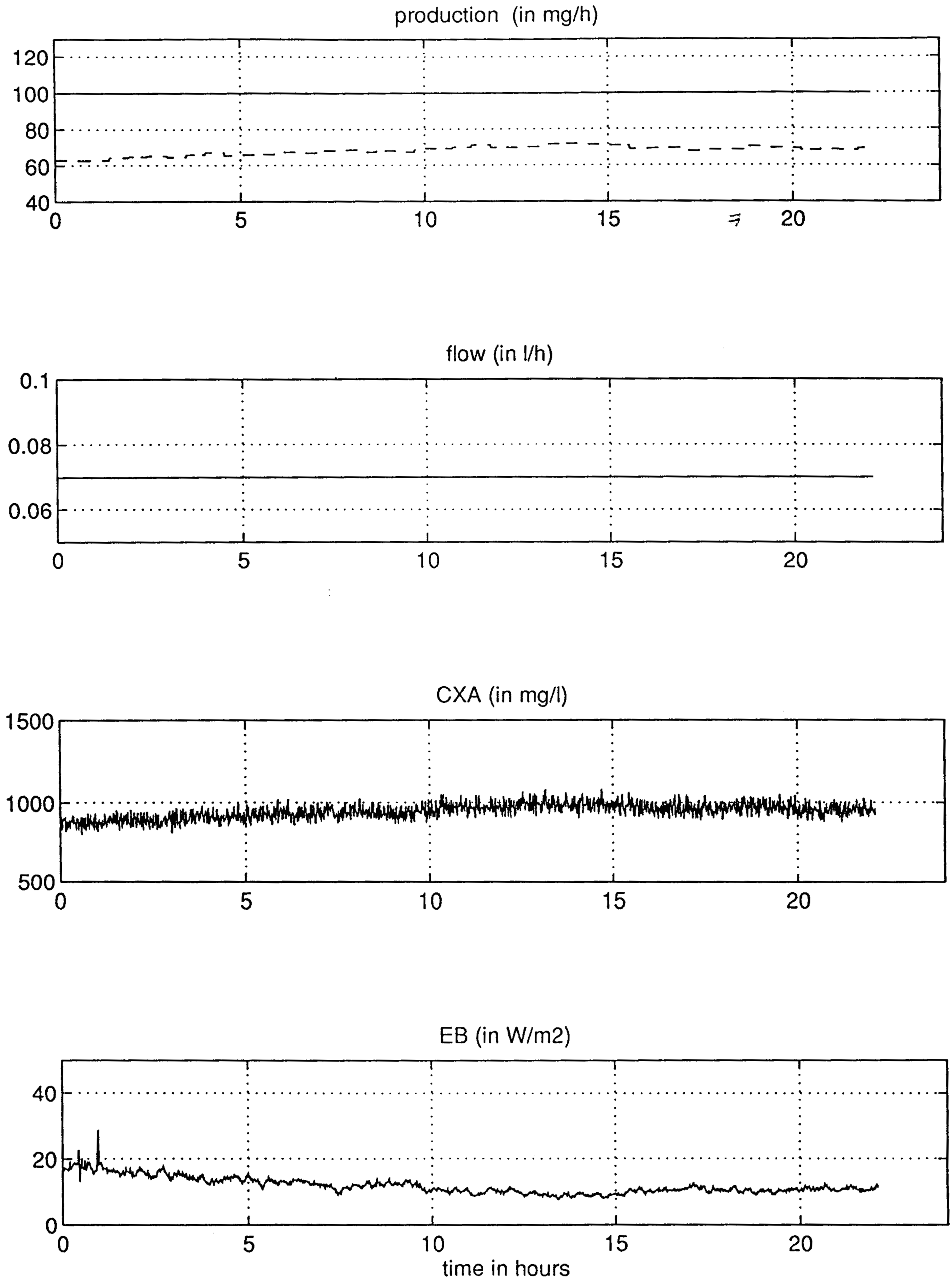


figure E7: experimental results of 14/03

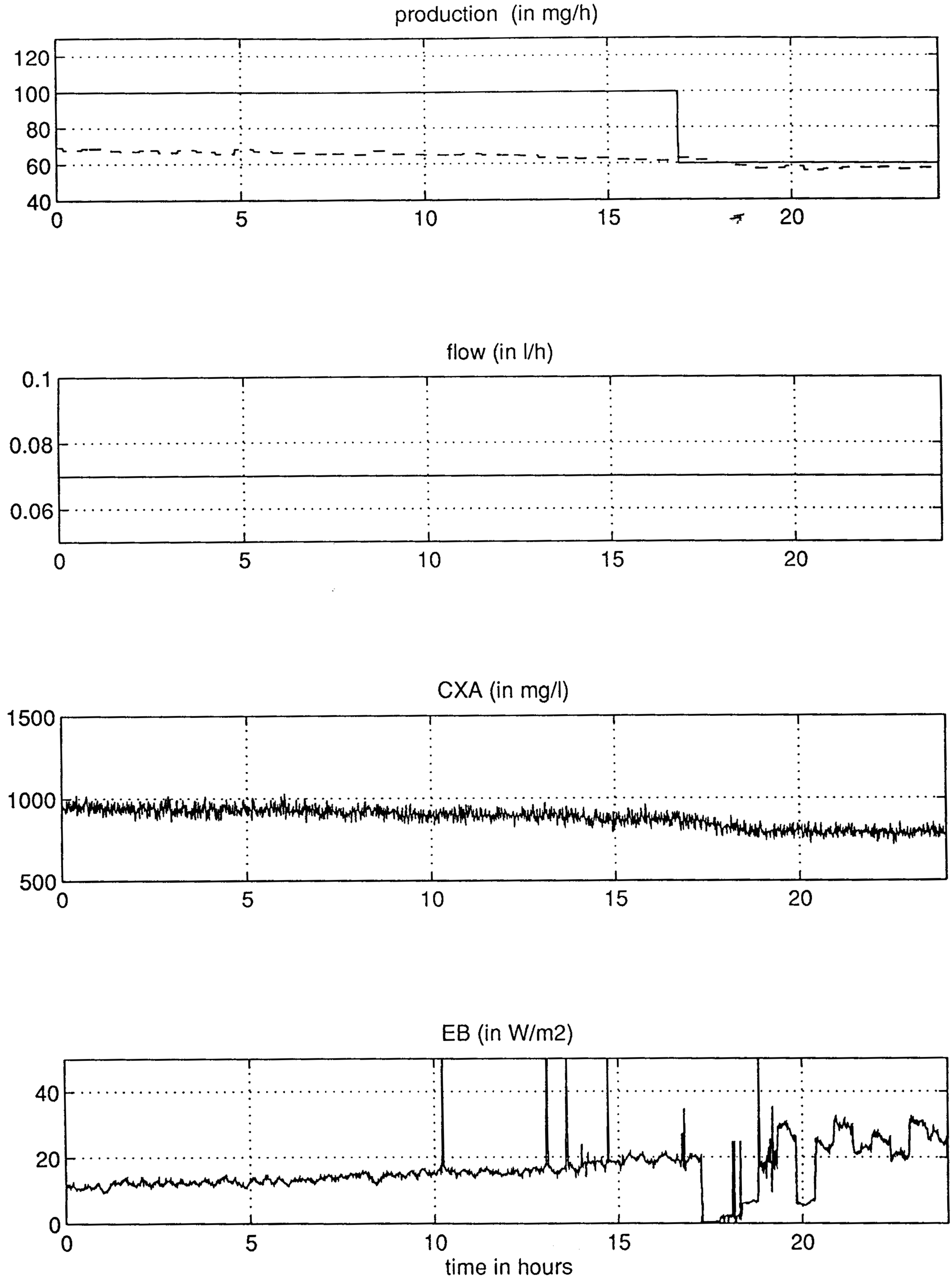


figure E8: experimental results of 15/03

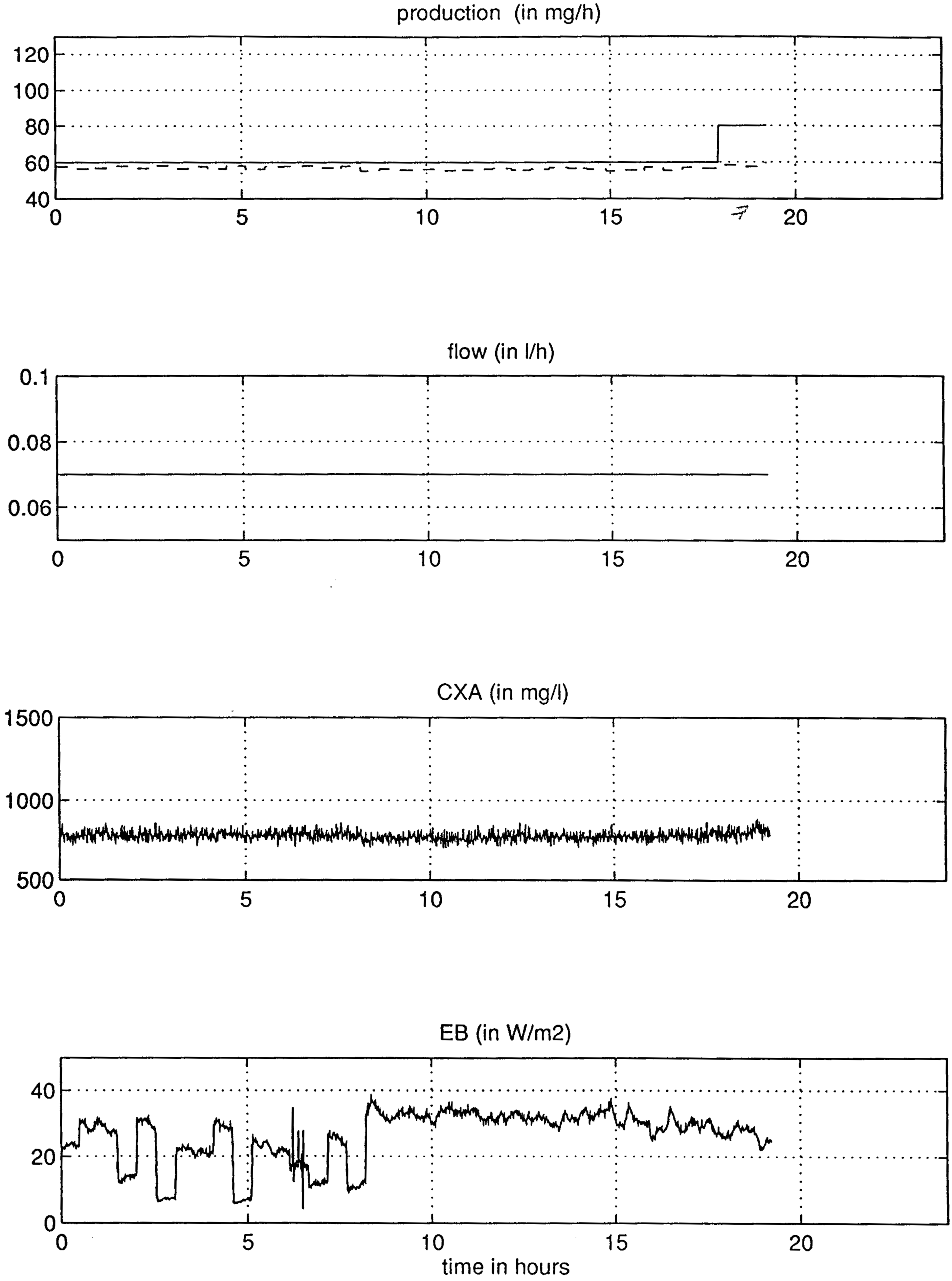


figure E9: experimental results of 16/03

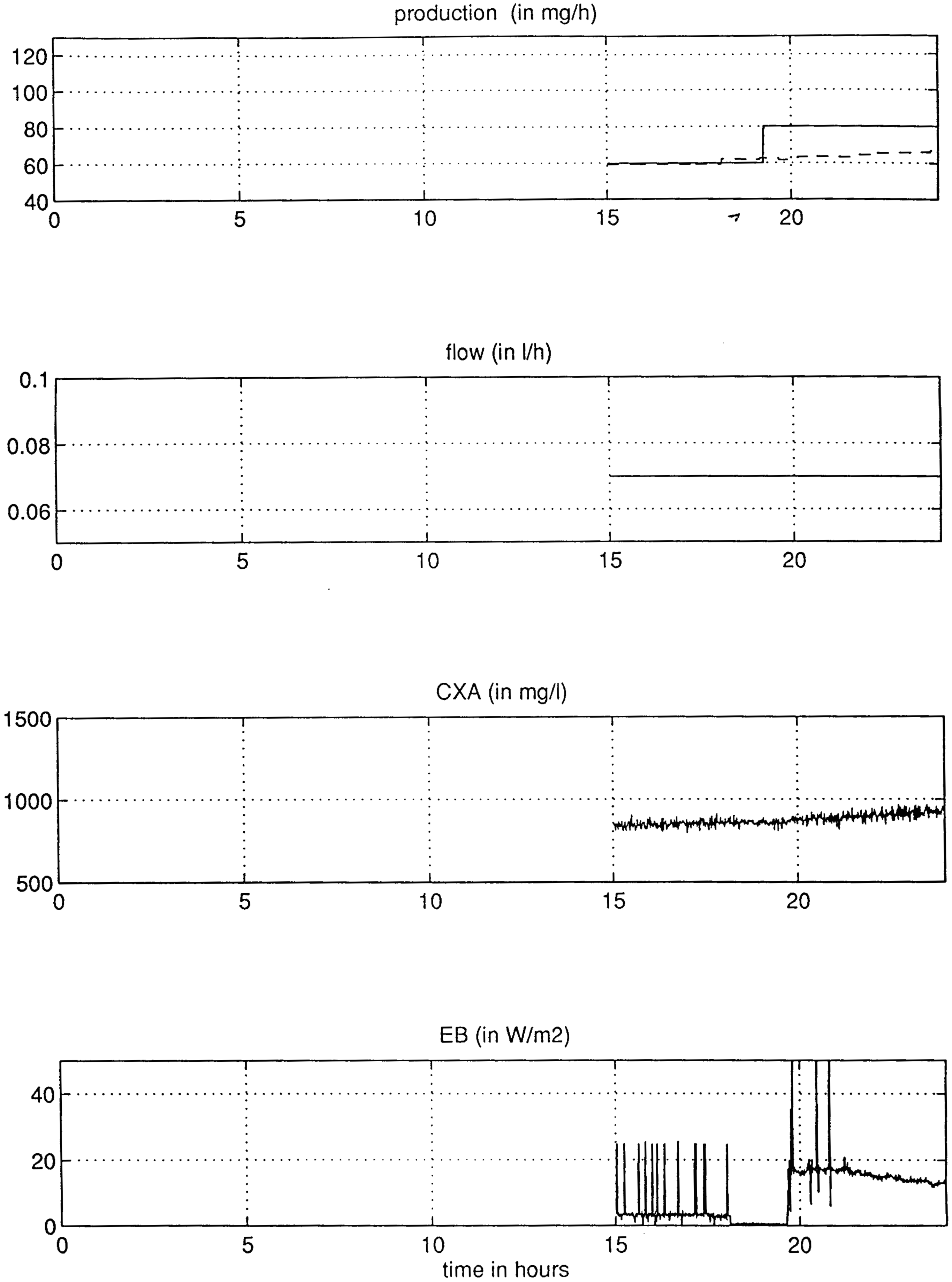


figure E10: experimental results of 18/03

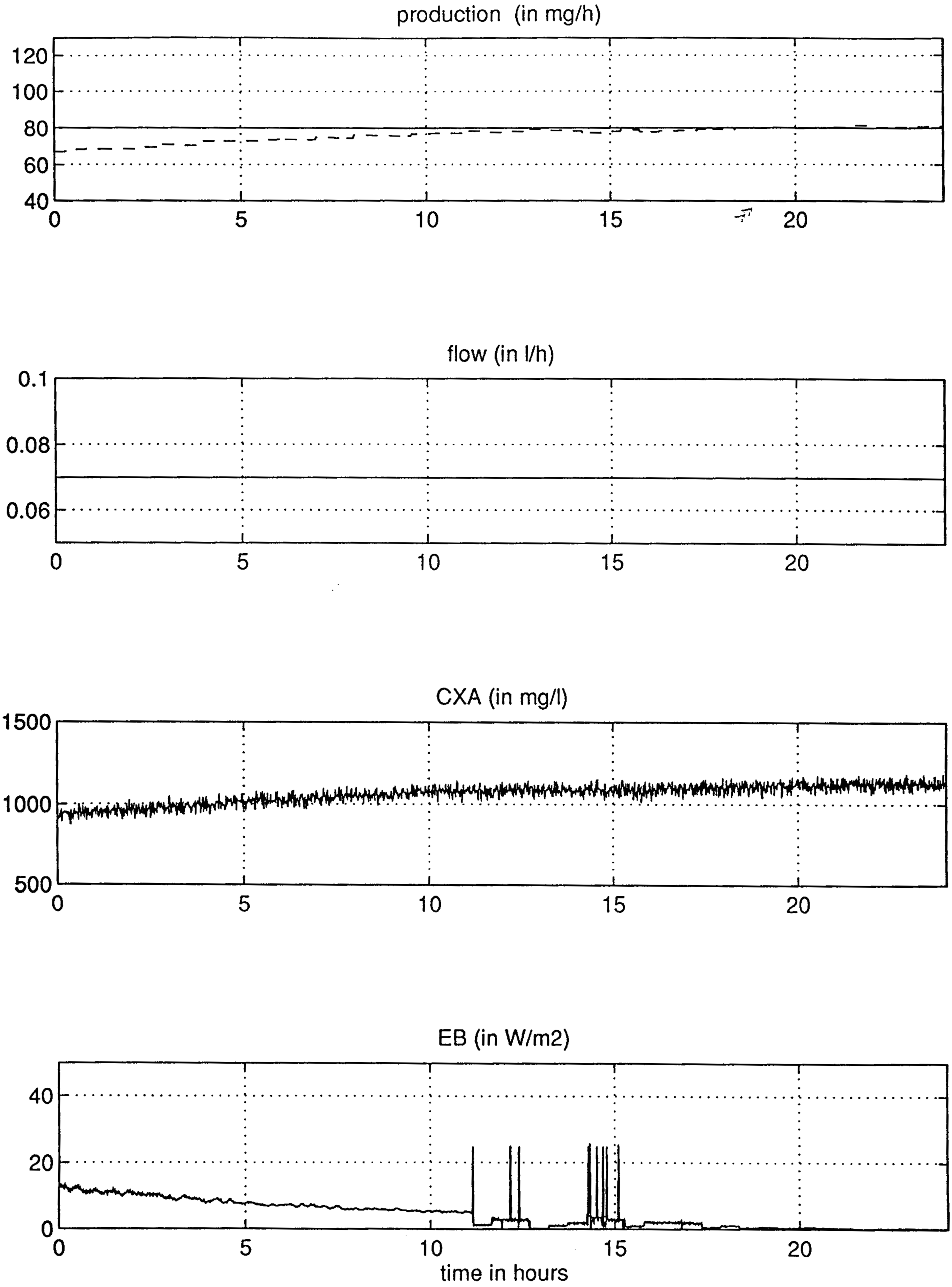
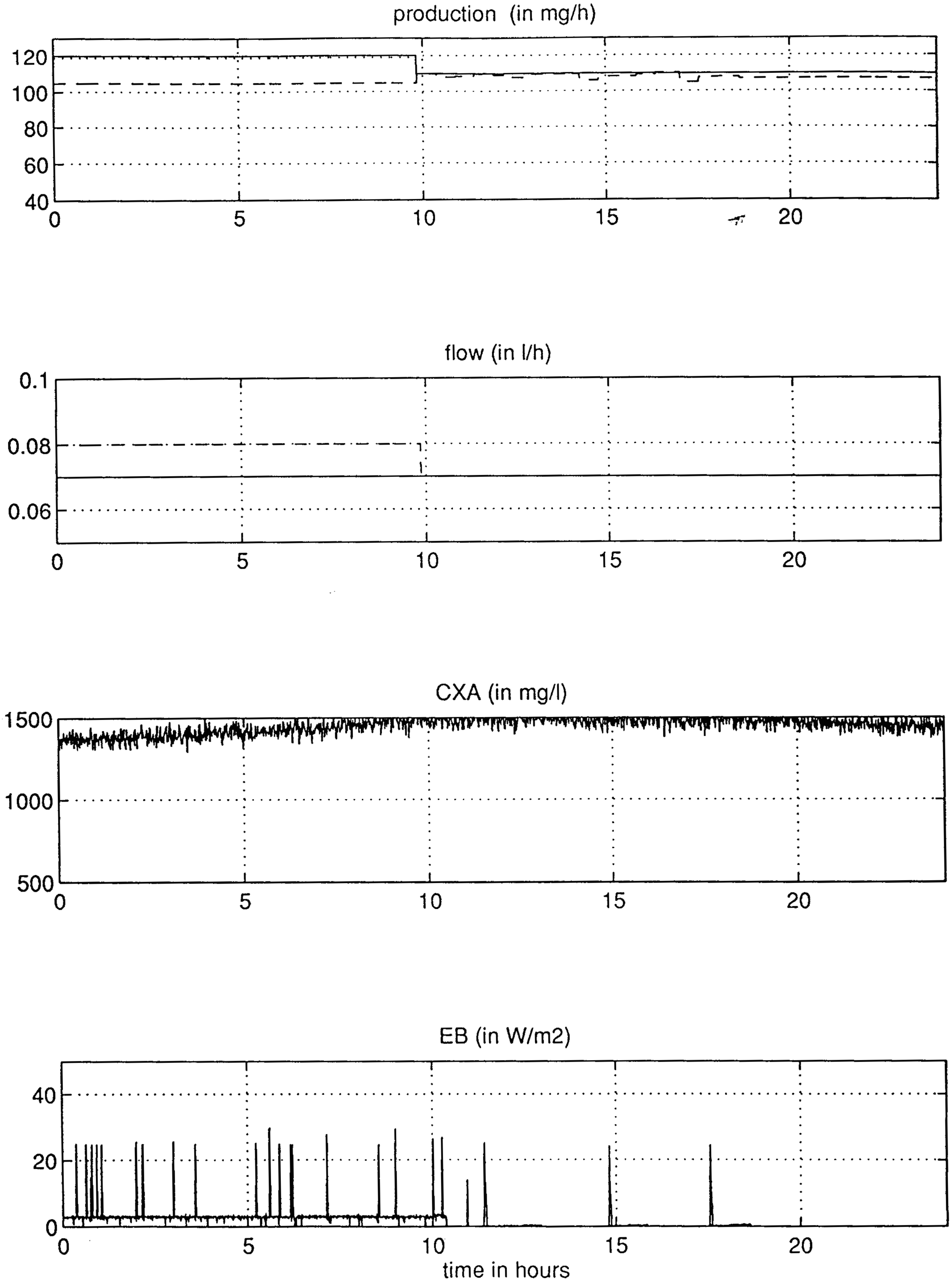


figure E11: experimental results of 19/03

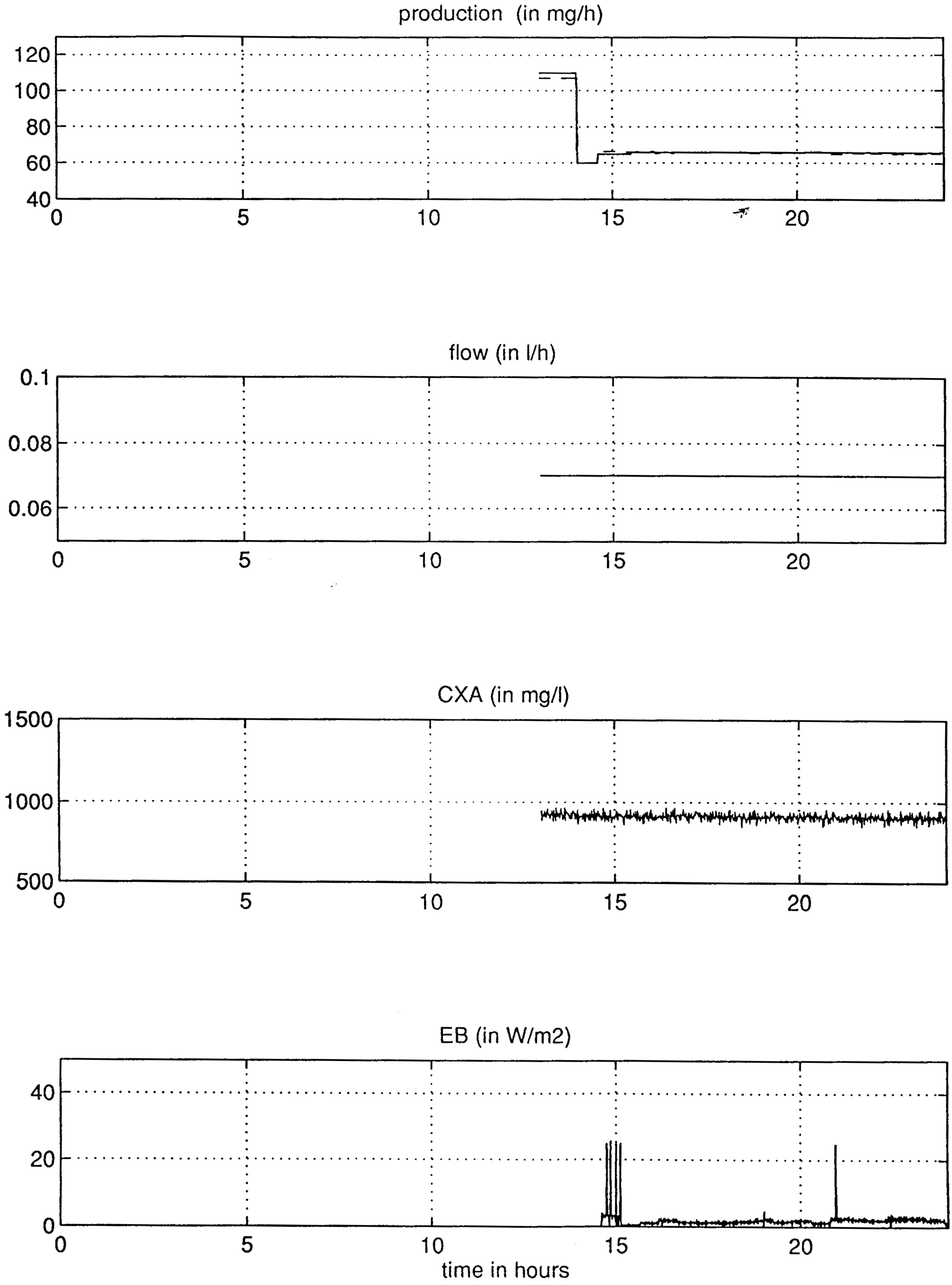

production (in mg/h)

flow (in l/h)

CXA (in mg/l)

EB (in W/m2)

figure E12: experimental results of 24/03



production (in mg/h)

flow (in l/h)

CXA (in mg/l)

EB (in W/m2)

figure E13: experimental results of 27/03

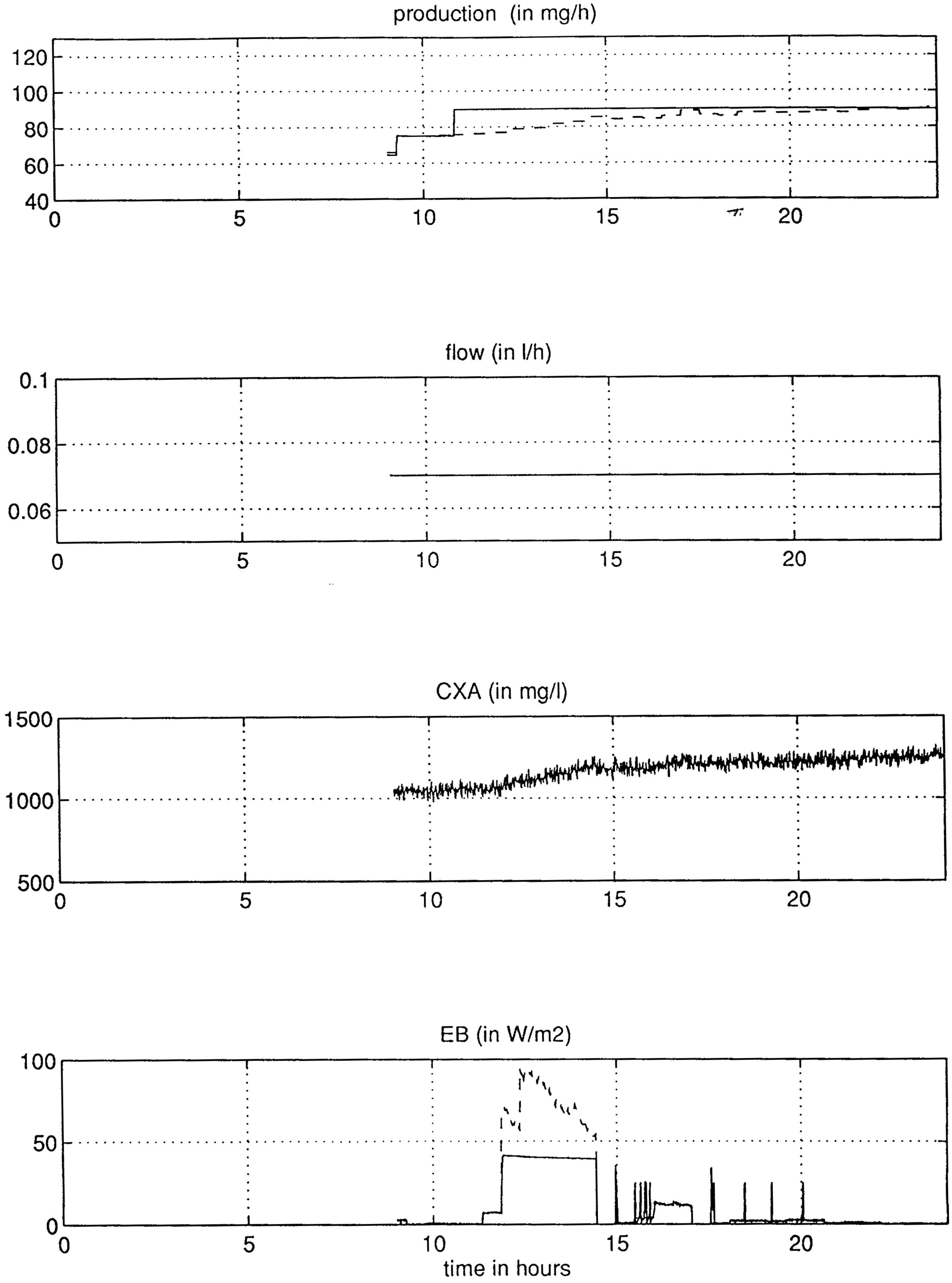


figure E14: experimental results of 29/03

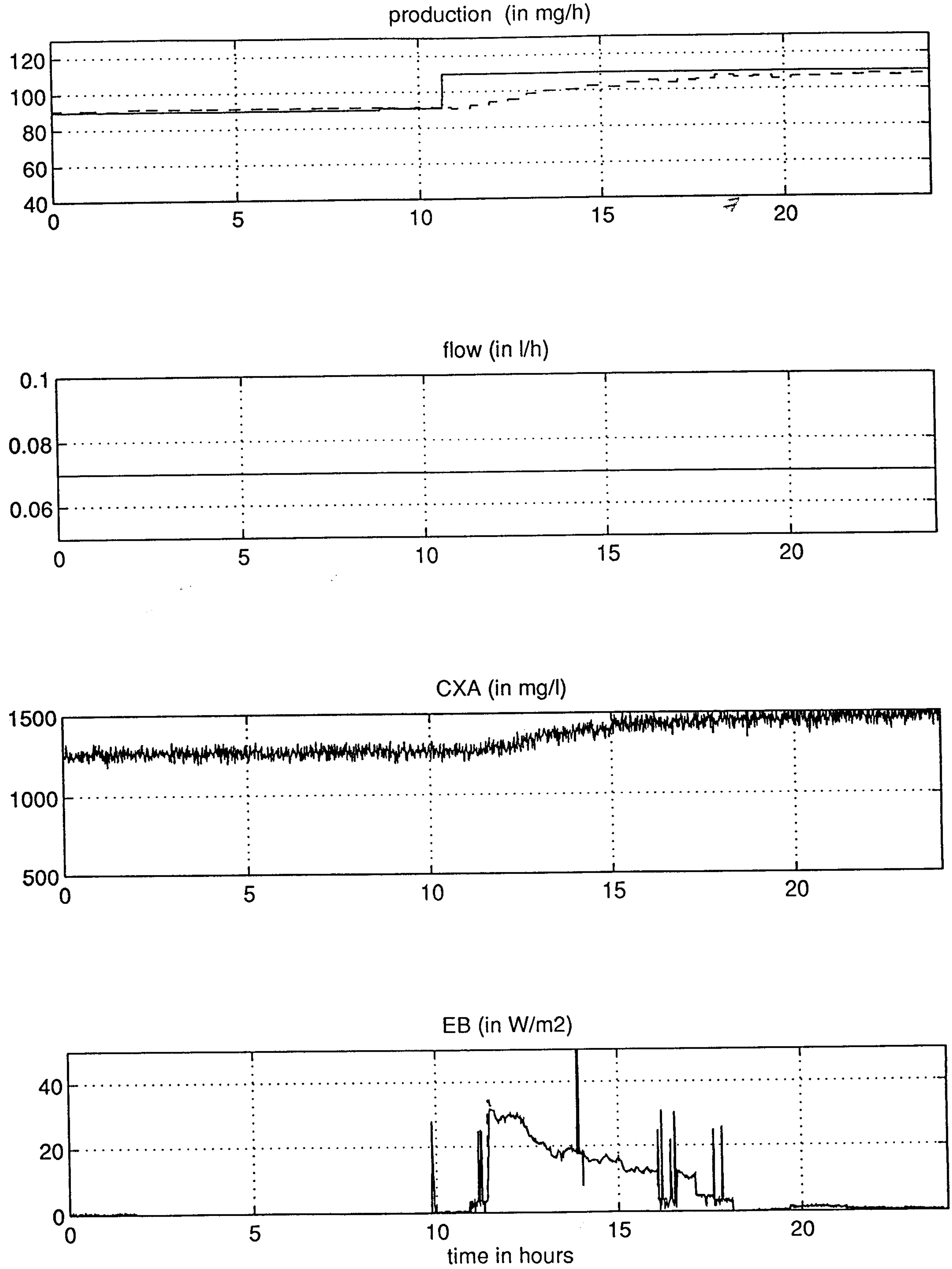


figure E15: experimental results of 30/03

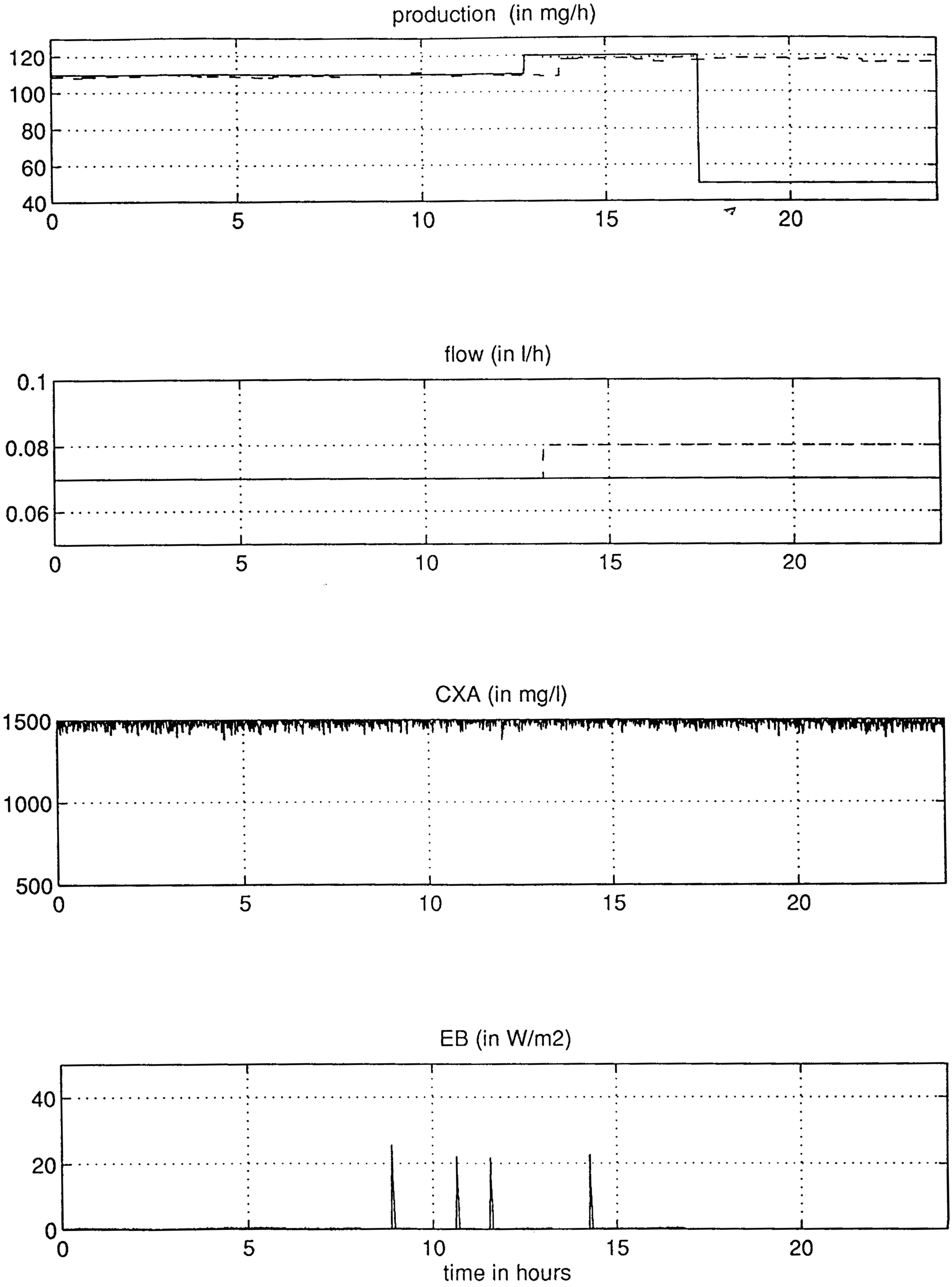


figure E16: experimental results of 31/03

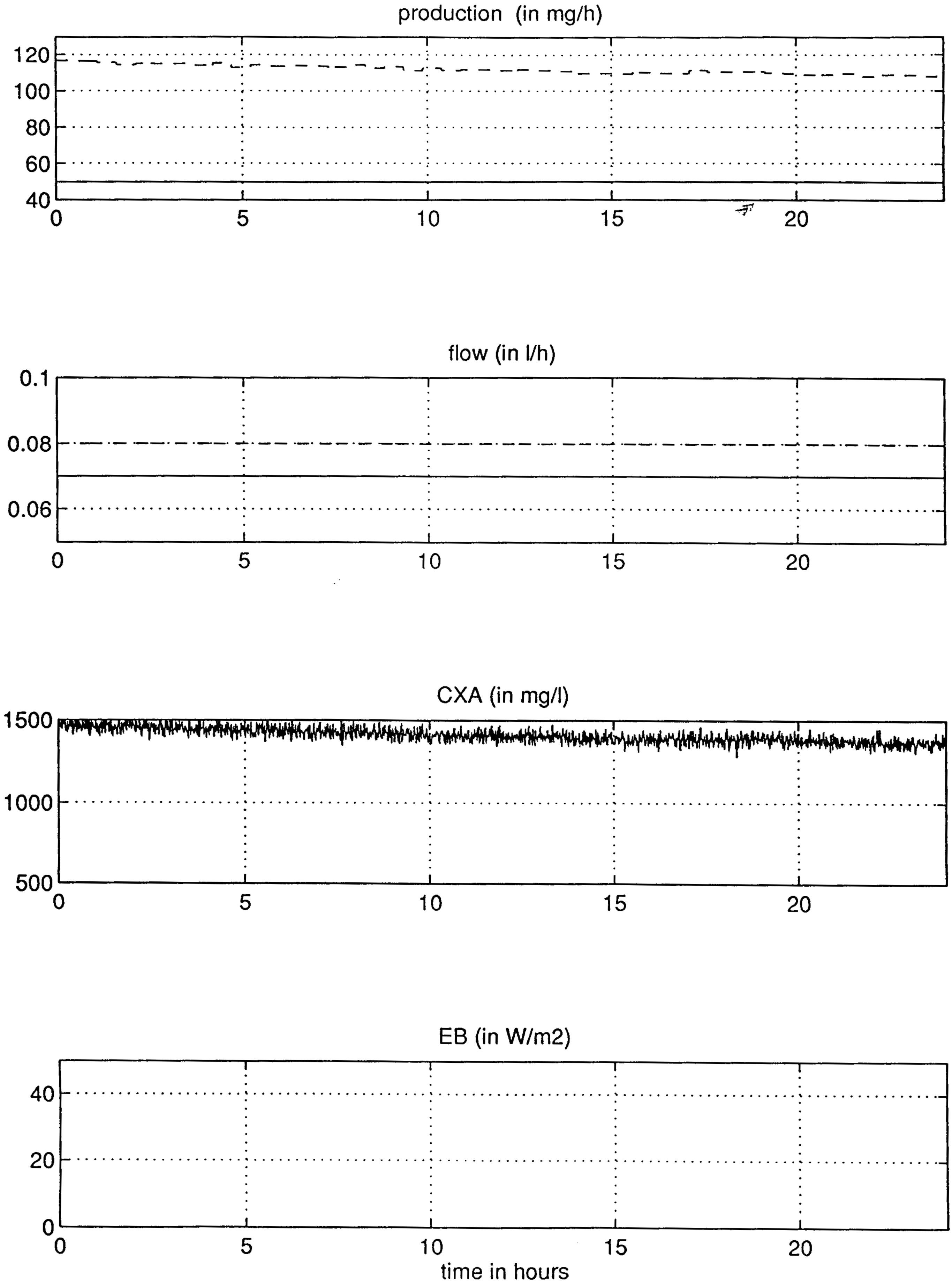


figure E17: experimental results of 01/04

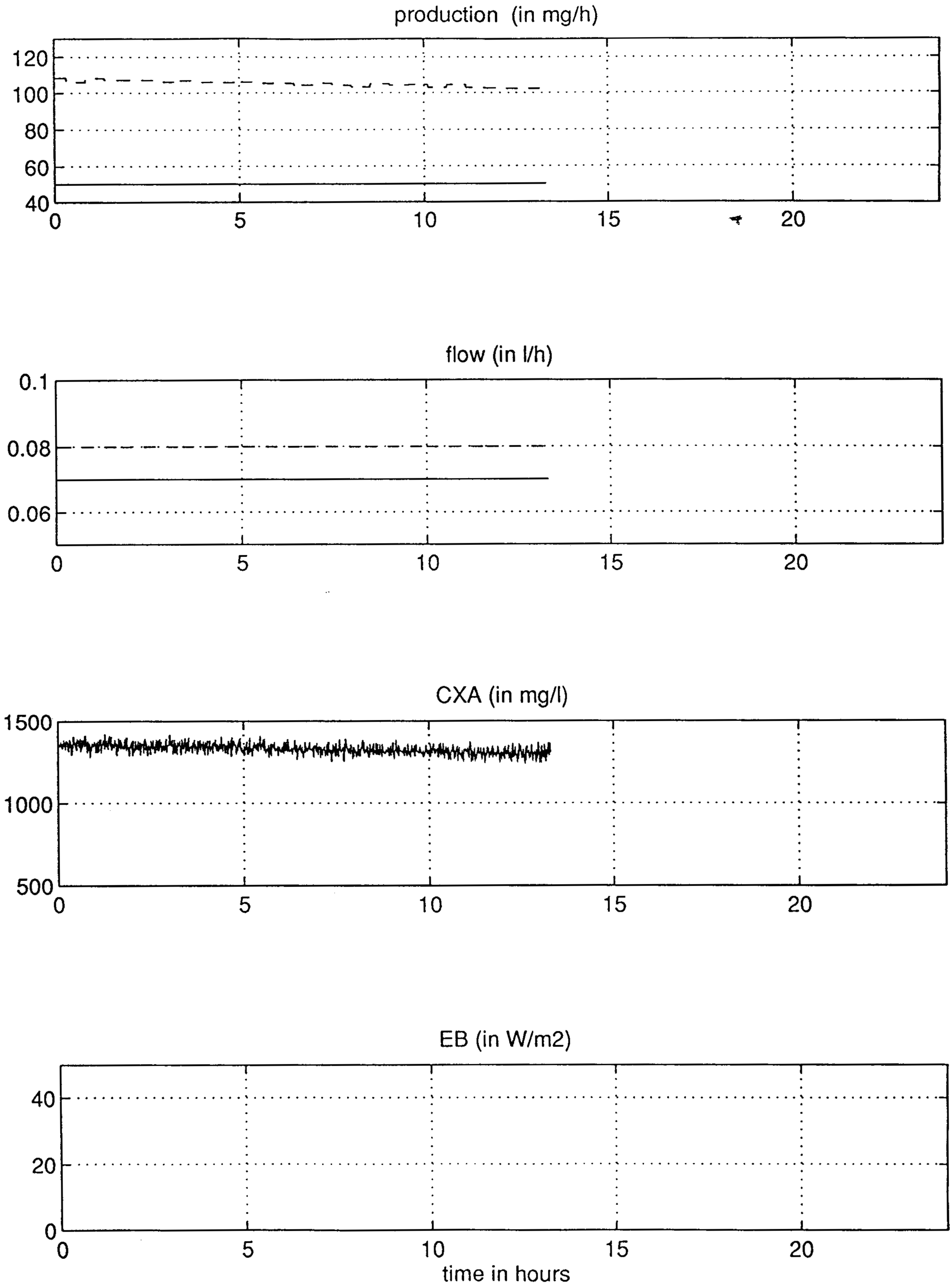


figure E18: experimental results of 02/04

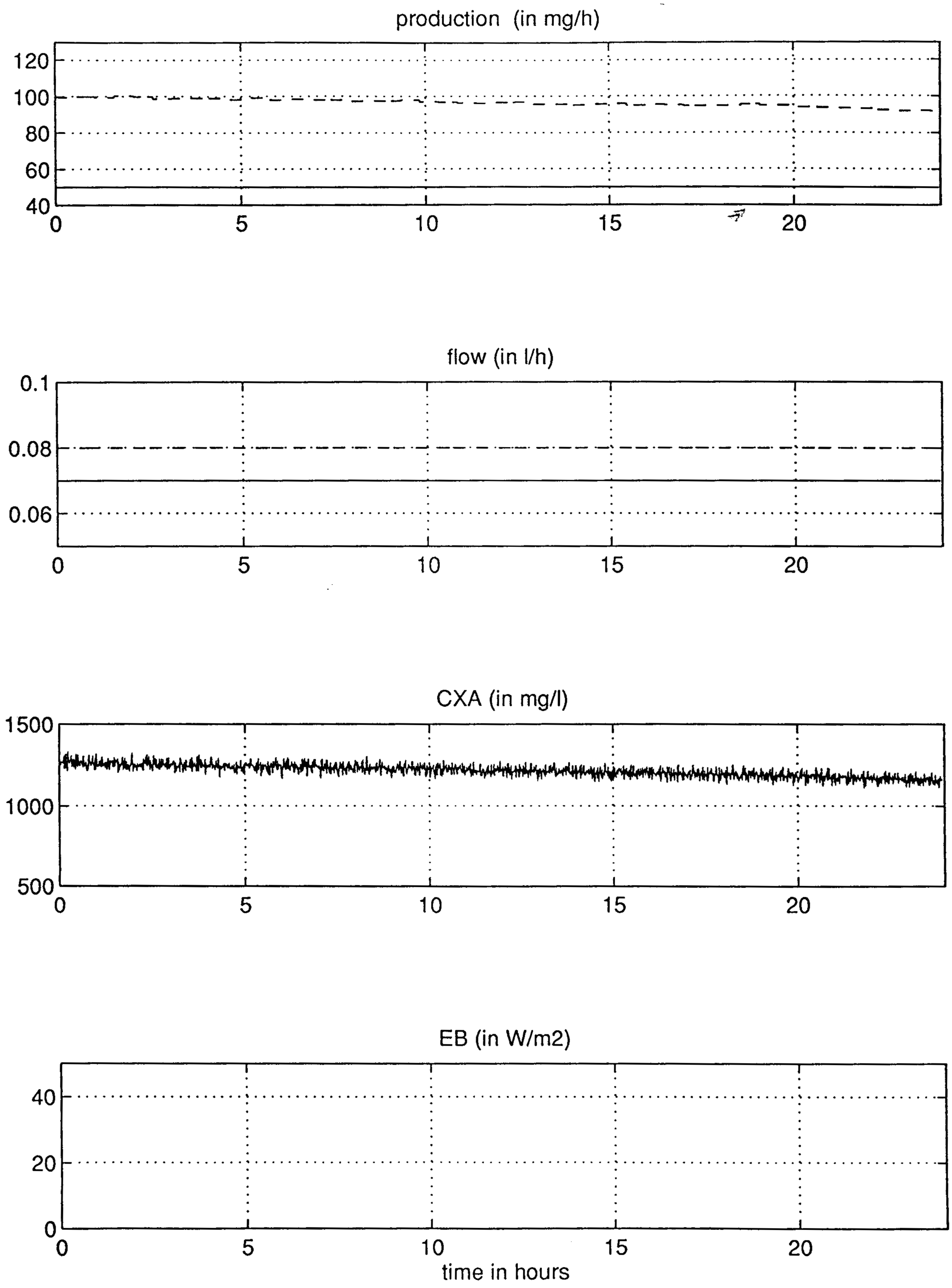


figure E19: experimental results of 03/04

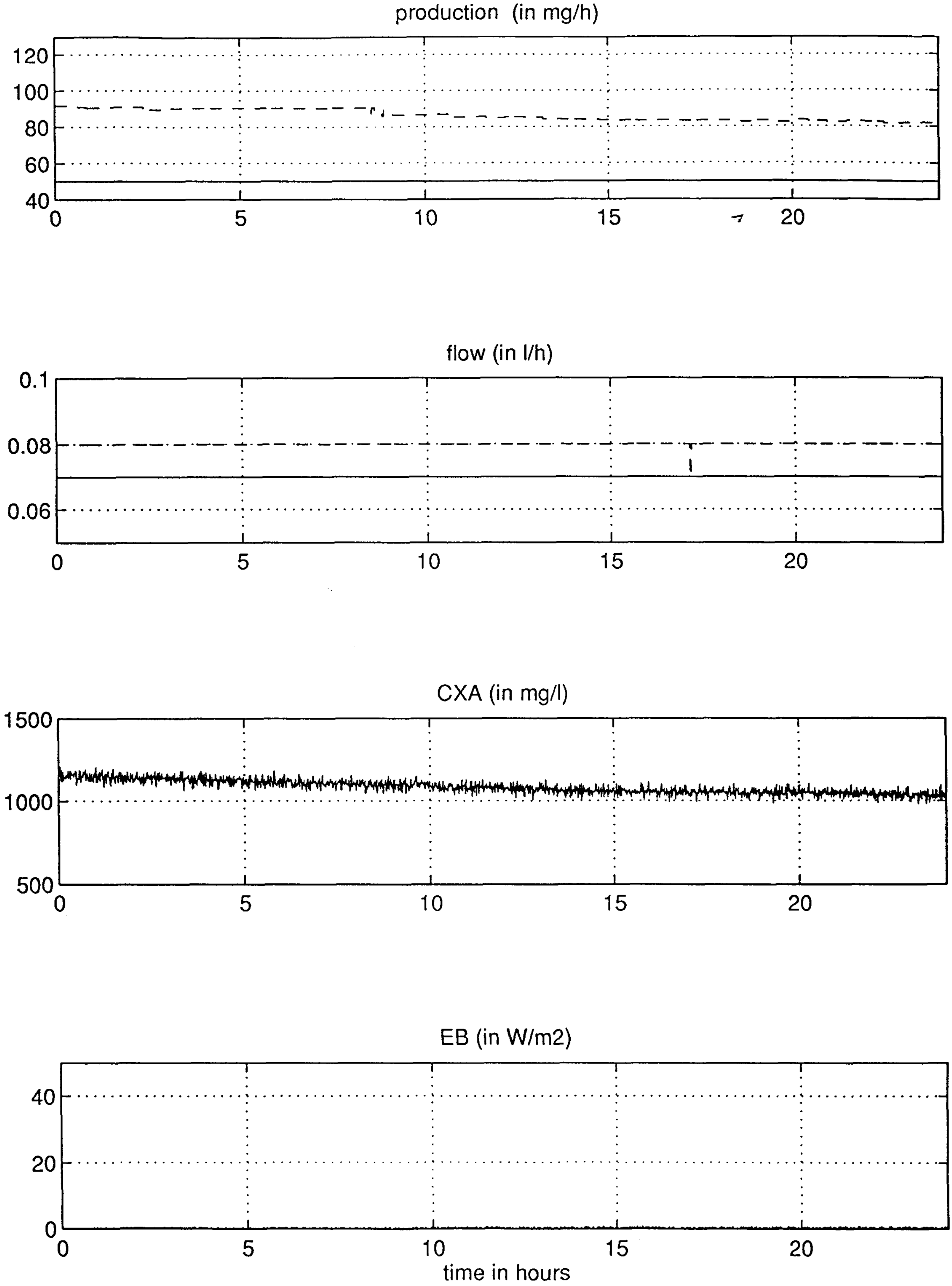


figure E20: experimental results of 04/04

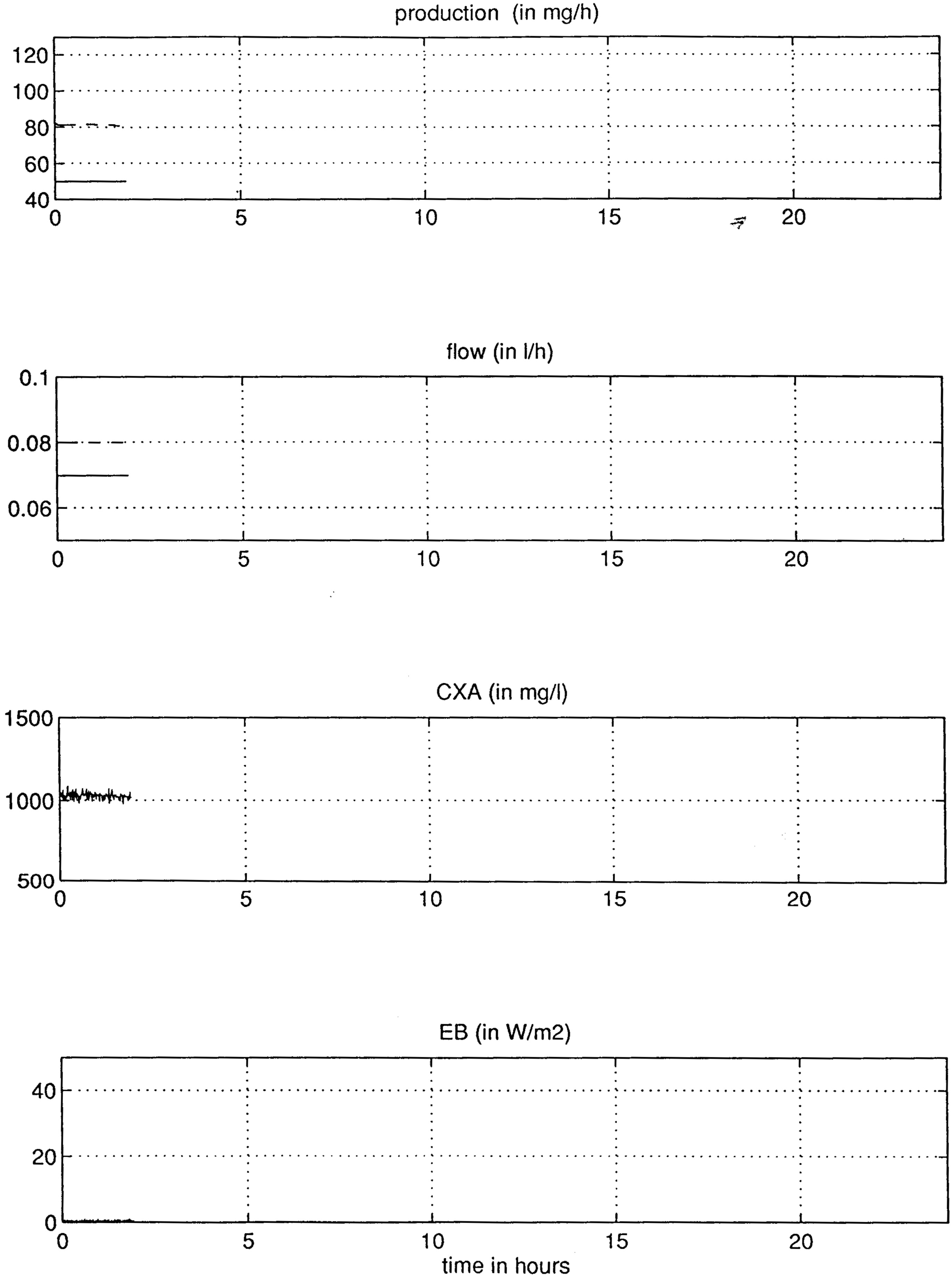


figure E21: experimental results of 05/04

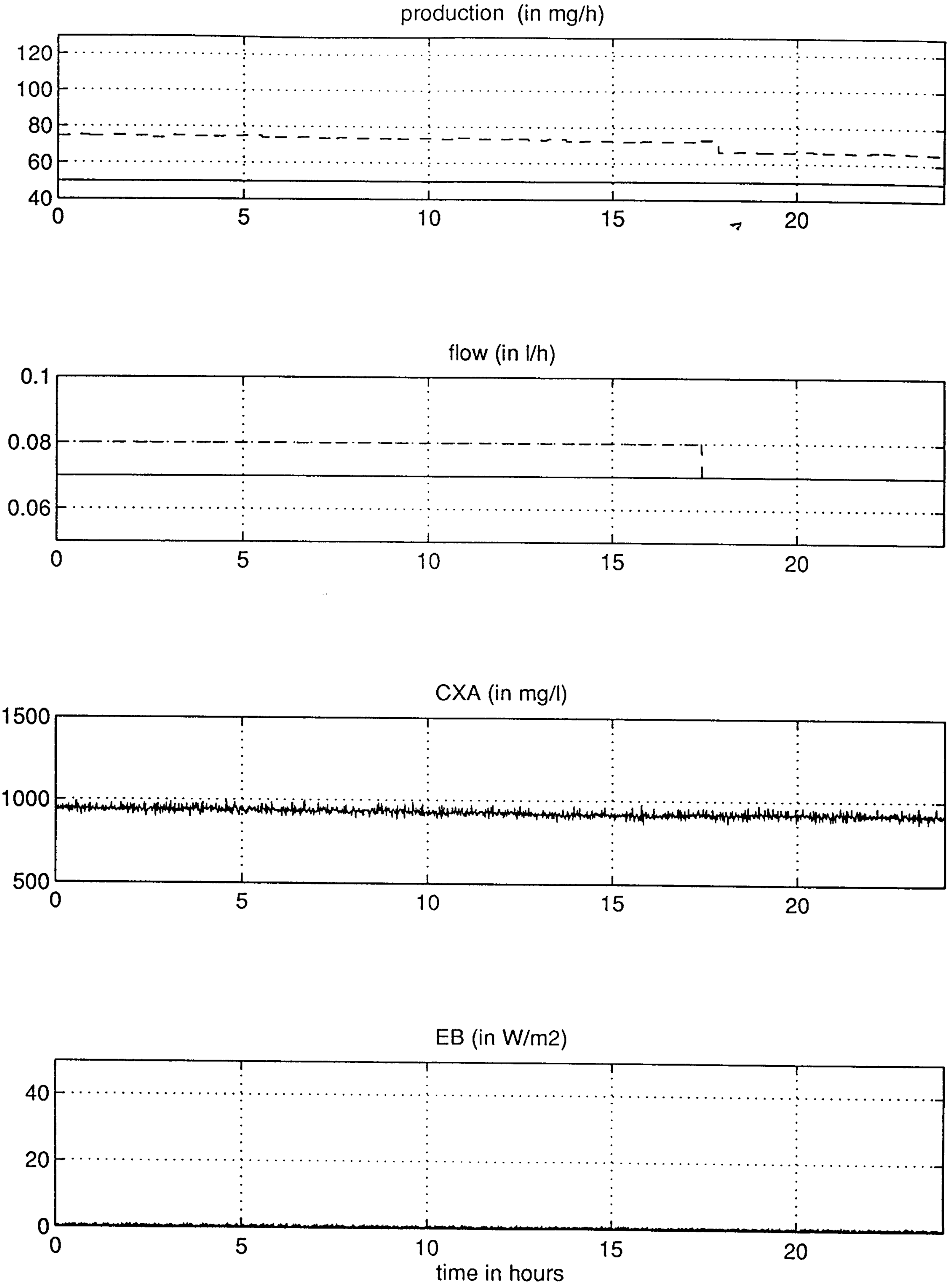


figure E22: experimental results of 06/04

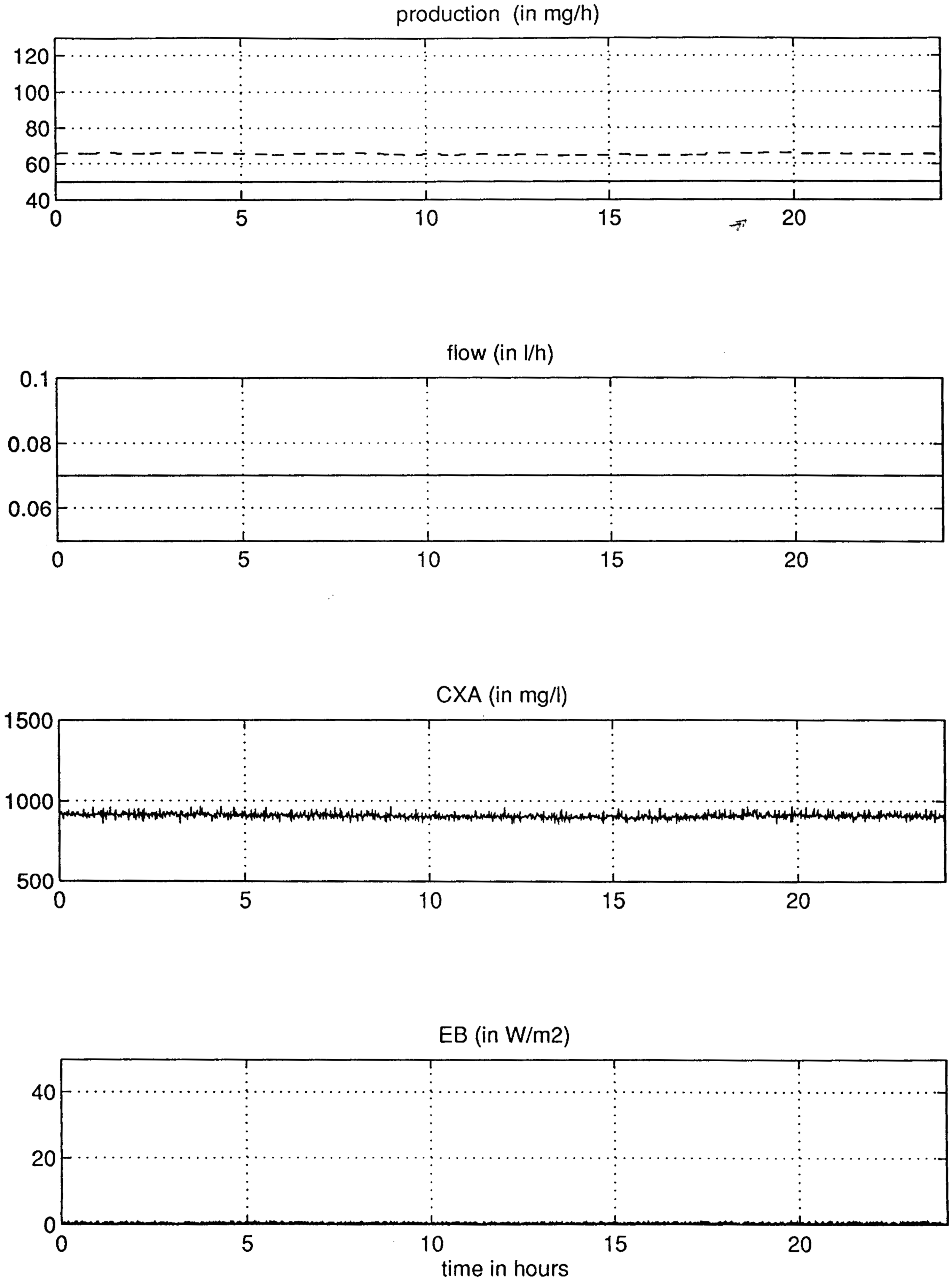


figure E23: experimental results of 07/04

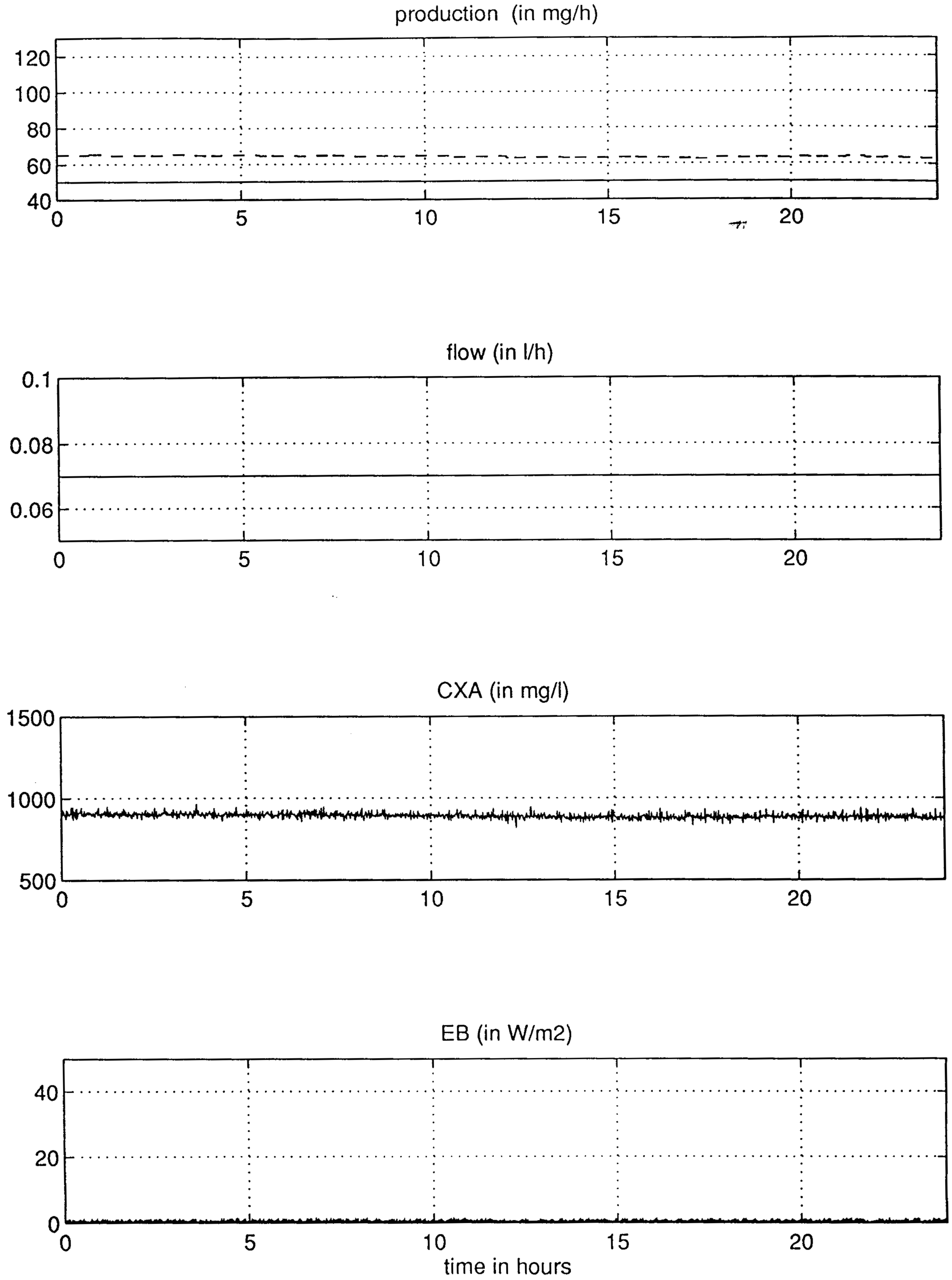


figure E24: experimental results of 08/04

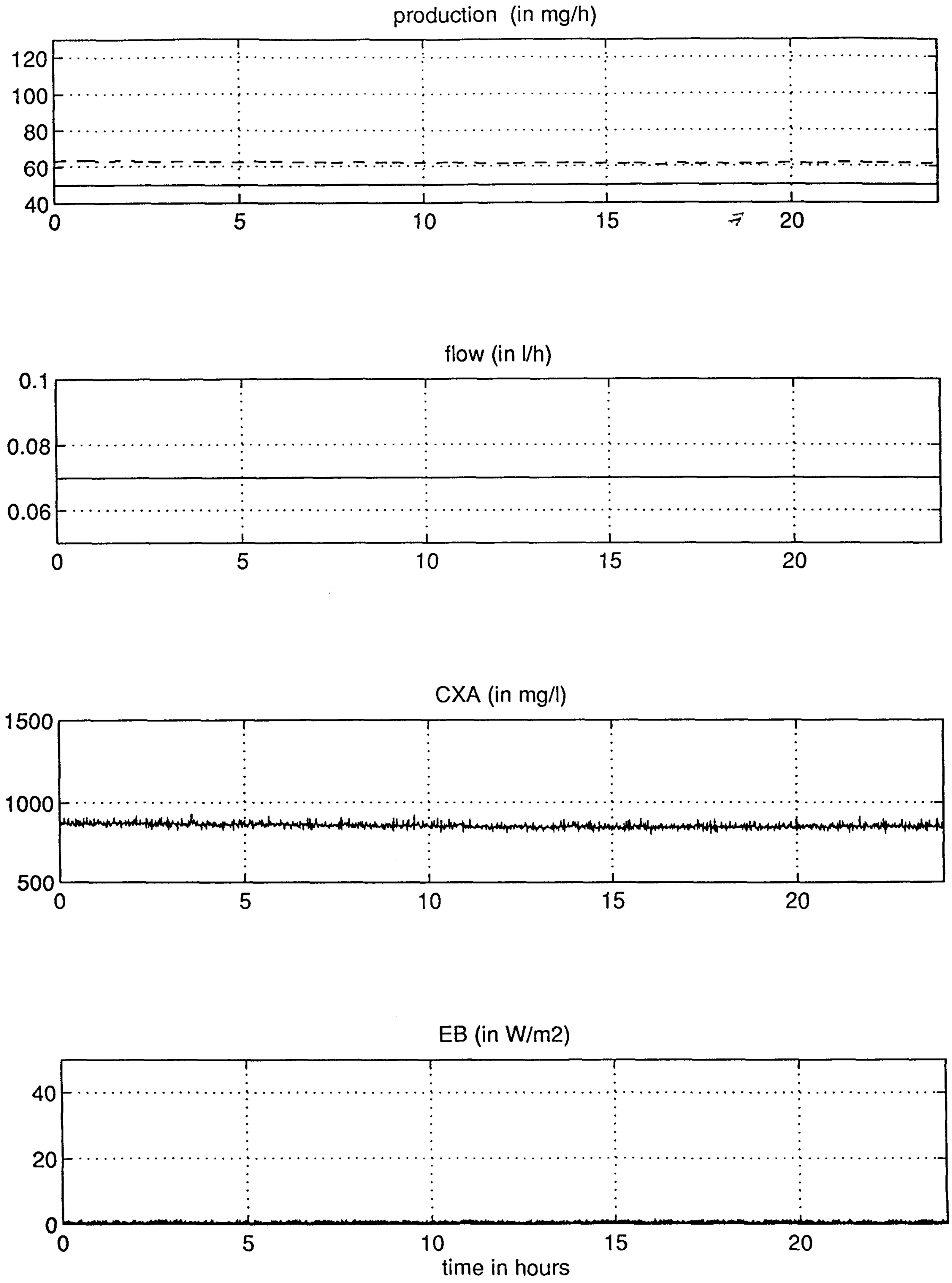


figure E25: experimental results of 09/04

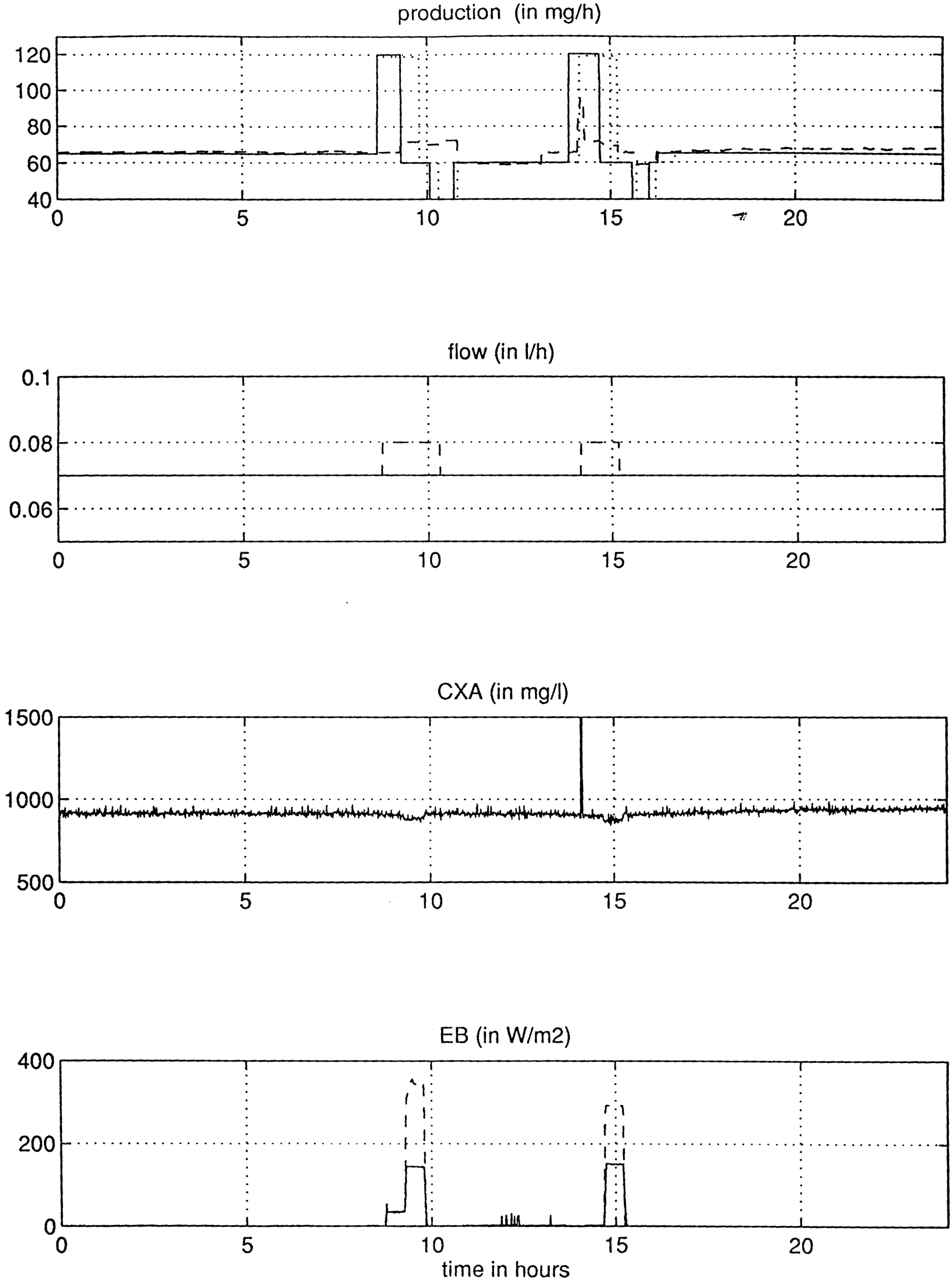


figure E26: experimental results of 12/04

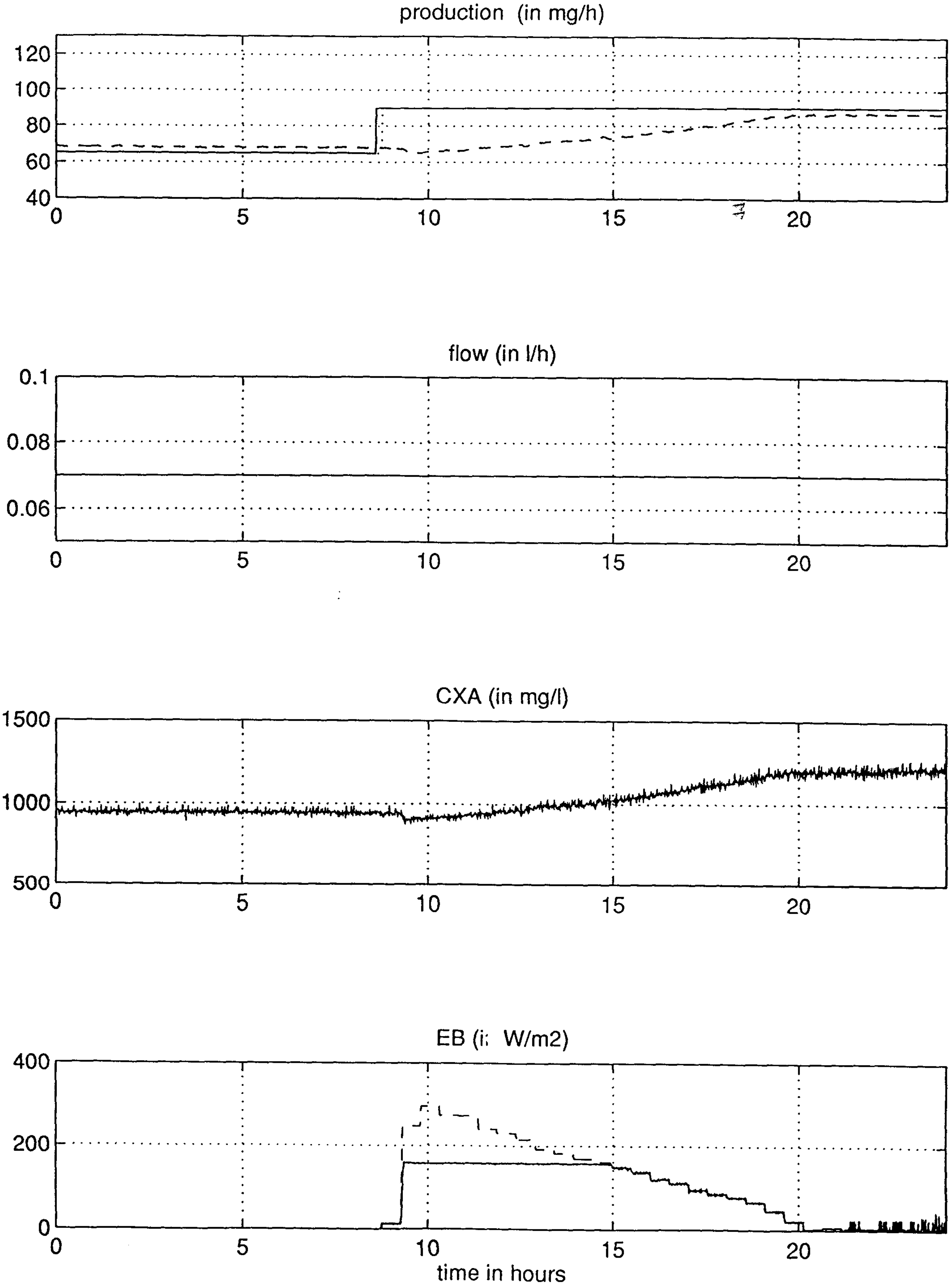


figure E27: experimental results of 13/04