143

JAERI-DATA-CODE-99-004

JAERI-Data/Code
99-004

FERMILAB
MAY 17 1999
LIBRARY

DEVELOPMENT OF DYNAMIC SIMULATION CODE FOR FUEL CYCLE OF FUSION REACTOR

Fermilab Library
0 1190 0062405 8

February 1999

NAME
RETURN TO FERMILAB LIBRARY

Isao AOKI, Yasushi SEKI, Makoto SASAKI*,
Kiyonori SHINTANI* and Yeong-Chan KIM*

日本原子力研究所
Japan Atomic Energy Research Institute

本レポートは，日本原子力研究所が不定期に公刊している研究報告書です。
入手の問合わせは，日本原子力研究所研究情報部研究情報課（〒319-1195　茨城県那珂郡東海村）あて，お申し越しください。なお，このほかに財団法人原子力弘済会資料センター（〒319-1195　茨城県那珂郡東海村日本原子力研究所内）で複写による実費頒布をおこなっております。

This report is issued irregularly.
Inquiries about availability of the reports should be addressed to Research Information Division, Department of Intellectual Resources, Japan Atomic Energy Research Institute, Tokai-mura, Naka-gun, Ibaraki-ken, 319-1195, Japan.

© Japan Atomic Energy Research Institute, 1999

編集兼発行　　日本原子力研究所

JAERI-Data/Code 99-004

Development of Dynamic Simulation Code for Fuel Cycle of Fusion Reactor

Isao AOKI, Yasushi SEKI, Makoto SASAKI*,
Kiyonori SHINTANI* and Yeong-Chan KIM*

Department of Fusion Engineering Research
Naka Fusion Research Establishment
Japan Atomic Energy Research Institute
Naka-machi, Naka-gun, Ibaraki-ken

(Received January 13, 1999)

A dynamic simulation code for fuel cycle of a fusion experimental reactor has been developed. The code follows the fuel inventory change with time in the plasma chamber and the fuel cycle system during 2 days pulse operation cycles. The time dependence of the fuel inventory distribution is evaluated considering the fuel burn and exhaust in the plasma chamber, purification and supply functions. For each subsystem of the plasma chamber and the fuel cycle system, the fuel inventory equation is written based on the equation of state considering the fuel burn and the function of exhaust, purification, and supply. The processing constants of subsystem for steady states were taken from the values in the ITER Conceptual Design Activity (CDA) report. Using this code, the time dependence of the fuel supply and inventory depending on the burn state and subsystem processing functions are shown.

Keywords: Simulation Code, Fuel Cycle, Fusion Experimental Reactor, Tritium Inventory, Pulse Operation, ITER CDA

---

* The Japan Research Institute, Limited

JAERI-Data/Code 99-004

核融合炉燃料循環系動特性コードの開発

日本原子力研究所那珂研究所核融合工学部
青木　功・関　泰・佐々木　誠*
新谷　清憲*・Yeong-Chan KIM*

(1999年1月13日受理)

核融合実験炉の燃料循環のシミュレーションコードを作成した。本コードは、パルス運転時のプラズマチェンバ及び燃料循環系内に分布する燃料の時間変化を追跡する。プラズマチェンバ及び燃料循環系における燃料の燃焼、排気、精製、供給の機能を時間当たりの処理量に着目し、その時間変化を追跡した。プラズマチェンバ及び燃料循環系各サブシステム毎に状態方程式と出力方程式を定め、燃料の燃焼、排気、精製、供給の機能をモデル化し、時間に関し定常となるサブシステムの常数は、ＩＴＥＲの概念設計書に依拠した。本コードを用いて、燃焼状態と燃料循環系サブシステムの処理機能とに依存する供給量の時間変化と、滞留量の時間変化を示した。

那珂研究所：〒311－0193　茨城県那珂郡那珂町向山８０１－１
＊㈱日本総合研究所

Contents

1. Introduction 1
2. Development of Dynamic Equation and Structure of Simulation Model 2
2.1 Development of Dynamic Equation 2
2.2 Structure of Simulation Model 3
2.3 The System Matrices 4
3. Simulation of Fuel Flow Cycle Model 5
3.1 Simulation Model in Subsystem and Simulation Results 5
3.1.1 Plasma Chamber 5
3.1.2 Vacuum Pump Subsystem 13
3.1.3 Fuel Clean-up Subsystem 18
3.1.4 Isotope Separation Subsystem 30
3.1.5 Neutral Beam Injection Subsystem 37
3.1.6 Pellet Injector Subsystem 43
3.1.7 Gas Puffer Subsystem 48
3.1.8 Storage and Supply Subsystem 51
3.2 Discussions 53
4. Summary 58
Acknowledgment 58
References 59
Appendix Source List 60

目　　次

1. はじめに …… 1
2. 動特性モデルの開発とシミュレーションモデルの構成 …… 2
2.1 動特性モデル開発の方針 …… 2
2.2 シミュレーションモデルの構成 …… 3
2.3 システムマトリックス …… 4
3. 燃料循環系のシミュレーション …… 5
3.1 シミュレーションモデルとシミュレーション結果 …… 5
3.1.1 プラズマチェンバ …… 5
3.1.2 真空排気サブシステム …… 13
3.1.3 燃料精製サブシステム …… 18
3.1.4 同位体分離サブシステム …… 30
3.1.5 中性粒子入射サブシステム …… 37
3.1.6 ペレット粒子入射サブシステム …… 43
3.1.7 ガスパフサブシステム …… 48
3.1.8 貯蔵・供給サブシステム …… 51
3.2 考　察 …… 53
4. まとめ …… 58
謝　辞 …… 58
参考文献 …… 59
付録　ソースリスト …… 60

1. Introduction

This report describes a dynamic simulation code for evaluating the time dependent tritium inventory and the flow rate in the fuel cycle subsystems of a tokamak fusion reactor. The control system of the fuel cycle system needs to integrate the individual process functions of the fuel cycle subsystems, which are related to each other through the input and output fuel flow. For such a complex control system, the reliable operation of the whole system needs to be demonstrated. In order to do so, engineering features of each subsystem need to be clarified. Such clarification is also required from the safety assurance of the operation of the subsystems. The engineering features of the subsystems can be clarified by observing the time and space variance of the subsystem responses. The observed values depend on the processing rate, spatial distribution, continuous or periodic operation mode of the subsystems.

A design of fuel cycle subsystems of a tokamak fusion reactor has been completed for the International Thermonuclear Experimental Reactor (ITER) design. The ITER Conceptual Design Activity (CDA) report [1] has been published in 1991. The report summarized the design requirements and the conceptual design for each of the principal subsystems and design options of the ITER fuel cycle. The ITER CDA fuel cycle design has been incorporated as the reference in the design studies to provide relevant data for the design requirements of the fuel cycle [2],[3]. In the references, several important insights into the behavior of the fuel cycle are provided. The time dependence of tritium inventory in fuel cycle subsystems was evaluated for a range of design operation parameters of the subsystems. One of the objectives was to reduce the tritium inventory as much as possible within the design constraints representing the various options in the subsystem design parameters. The tritium inventory determined by the type of operating conditions of the burning scenario was evaluated. In this approach, understanding transient system behavior incorporated in the fuel cycle subsystems operating modes and their associated scheduling is needed for predicting how the tritium inventory can be affected by the various fuel cycle subsystems behavior.

In this report, the fuel cycle is considered to be an integrated system of each subsystem functions. In order to understand the controllability of the

integrated system, time and space dependence of the fuel cycle subsystem responses to input fluctuation was observed. To meet the above objective, a fuel cycle dynamics simulation code which takes into account the response rate, spatial distribution and process function of subsystems was developed. The processing constants of subsystems and proposed nominal ITER pulse operation scenario were adopted from the ITER-CDA report [1].

2. Development of Dynamic Equation and Structure of Simulation Model

2.1 Development of Dynamic Equation

The important feature of the state-space approach over the time domain consideration is that both time-varying and time-invariant linear or nonlinear systems can be treated systematically. The system with the state-space description is expressed with the time-invariant constants whose values is definitely determined over the time in the proposed nominal ITER pulse operation scenario in ITER CDA Design. Introducing the state vector and excitation vector, the description of state-space is formulated by using the system matrices according to the design description in the ITER CDA. The state vector denotes the state variables, Tritium, Deuterium, Protium, each of which represents the controllability in the change of time governing the fuel processing cycle. The excitation vector denotes the derivatives of the proposed nominal ITER pulse operation scenario. The time-invariance system whose input output response are governed by linear ordinary differential equations is established as following;

$$\frac{d\mathbf{x}(t)}{dt} = \mathbf{A}\mathbf{x}(t) + \mathbf{B}\mathbf{u}(t) \tag{2.1}$$

$$\mathbf{y}(t) = \mathbf{C}\mathbf{x}(t) + \mathbf{D}\mathbf{u}(t) \tag{2.2}$$

,where $\mathbf{x}(t)$, $\mathbf{u}(t)$, $\mathbf{y}(t)$ denote state, excitation, and observation vector, respectively. $\mathbf{A}$, $\mathbf{B}$, $\mathbf{C}$, $\mathbf{D}$ in the equation (2.1) and (2.2) are system, control, observation, and transfer matrices, respectively. From the theory of ordinary differential equations, equation (2.1) has a unique solution given by

$$\mathbf{x}(t) = e^{\mathbf{A}t}\mathbf{x}(t_0) + \int_{t_0}^{t} e^{\mathbf{A}(t-\tau)}\mathbf{B}\mathbf{u}(\tau)d\tau \tag{2.3}$$

where $\mathbf{x}(t_0)$ is the initial state at the time t_0. It is clear from equation (2.3) that the state vector, $\mathbf{x}(t)$, in the state-space description are linear for the

initial state. The equation (2.3) denotes a fact that state vector, $\mathbf{x}(t)$, is completely observable because the solution is comprised of a term of the homogeneous equation, $e^{\mathbf{A}t}\mathbf{x}(t_0)$, and a term of the excitation, $e^{\mathbf{A}(t-\tau)}\mathbf{Bu}(\tau)$, convoluted over the observation time $t-\tau$. Then, the solution denotes that the state vector is completely controllable. A complete controllability is introduced from the fact that in the time-invariant linear system one controllability matrix is the transpose of the corresponding observability matrix [4]. The state-space approach in the time domain is to deal with the state-space description on the condition that the static dynamic behavior covering the fuel processing cycle is linear in the discrete-time systems. Discrete-time, $t+\Delta t, t$, is supposed to coincide in the time-invariant linear with the following periods of time to trace the time response for connecting process and process controllers over the entire processing cycle.

2.2 Structure of Simulation Model

The functions of a process control system that are implemented to the subsystems are burning, exhausting, purification, isotope separation, and storage and supply. Subsystems where the functions of a process control system are implemented are Plasma Chamber, Vacuum Pumping, Fuel Cleanup, Isotope Separation, Fuel Supply (Gas Puff, Pellet Injection, Neutral Beam Injection), and Storage and Supply. The state-space description is given by equations (2.1) and (2.2). Equation (2.1) describes the control process. Equation (2.2) describes the observation process. Equation (2.2) expresses that observation vector, $\mathbf{y}(t)$, not only depends on the state vector, $\mathbf{x}(t)$, but also depends directly on the excitation vector, $\mathbf{u}(t)$. Consequently, a variance in the time of $\mathbf{y}(t)$ denotes the change of inventories of the state variables over the observation time. Then, the state-space description of each subsystem was established to trace the change of the time of state vector whose components are Tritium, Deuterium, and Protium forced by the information of pulse operation scenario over entire processing cycle. Fig. 2.1 shows the established fuel flow cycle model.

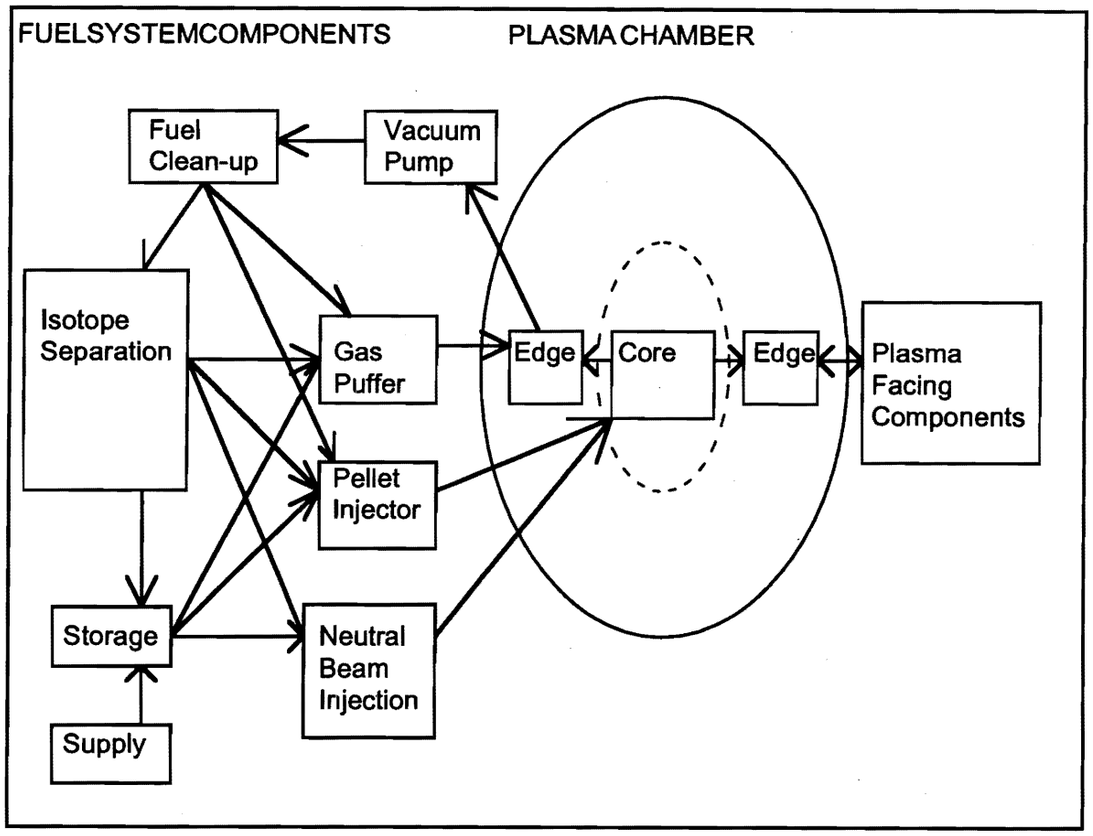


Fig. 2.1 Fuel flow cycle model

2.3 The System Matrices

The formations of time-invariant systems whose input-output responses are governed by linear ordinary differential equations were established. The state-space descriptions in the fuel processing cycles have been formulated in the linear ordinary differential equations by using the system matrices according to the design description reported in the ITER CDA Report [1]. The system matrices are determined by the time-invariant constants which describe the unit function of process controllers. To fully realize the state-space descriptions of the fuel processing cycles forced by the plasma chamber exhaust flows, the linear ordinary differential equations for subsystems where the unit function of process controllers are implemented are formulated. The state-space description expresses the problem of solving the simultaneous equations in a matrix. Hence to verify stability of the state-space description, one has to prove the existence of solution for the

variance of sate vector. The simultaneous equation gives a solution only if system matrix, $\mathbf{A}$, is nonsingular matrix. To verify this result an investigation procedure described in Fig. 2.2 was implicated and executed on each simulation step.

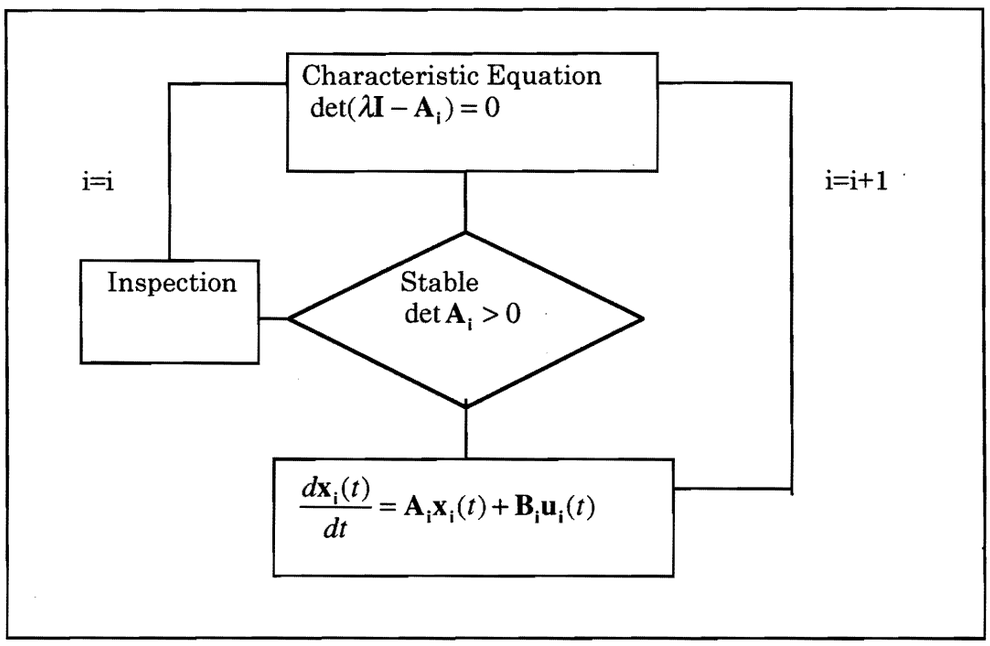


Fig. 2.2 A scheme of the investigation procedure i.

3. Simulation of Fuel Flow Cycle Model

3.1 Simulation Model in Subsystem and Simulation Results

3.1.1 Plasma Chamber

(1) Pulse Operation Scenario

The pulse operation history of fusion power must be consistent with the reference plasma operation scenario. The reference plasma operation scenario consists of burn and dwell times. Burn time follows three phases, heating, burn, and burn termination. Fig. 2.3 shows the nominal plasma operating cycle. The fuel processing cycle forced by the plasma chamber exhaust flow is simulated over the pulse operation cycles lasting 2 days.

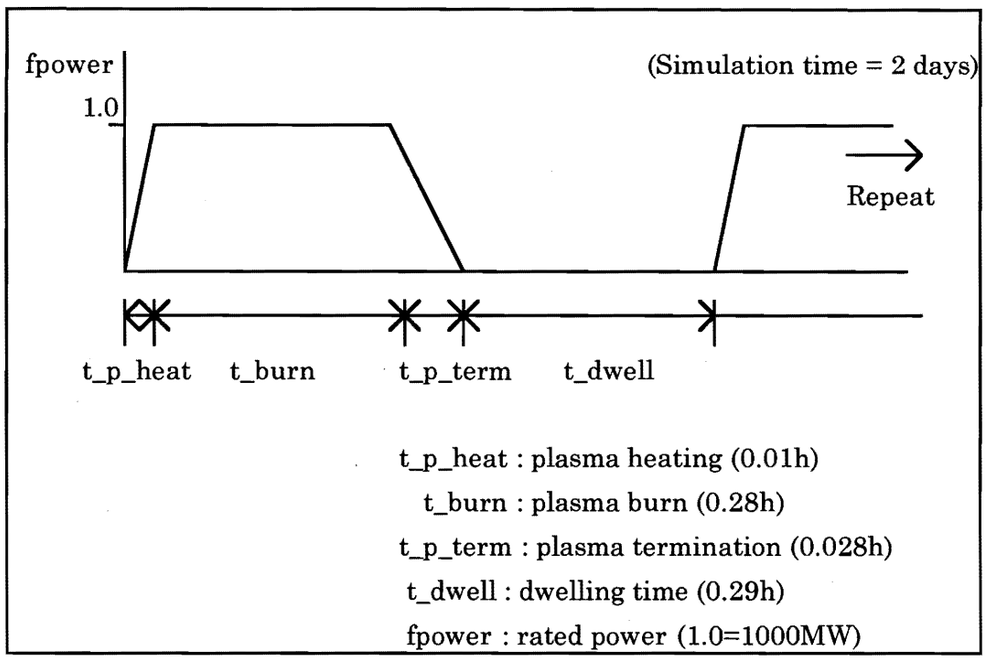


Fig. 2.3 Nominal plasma operating cycle

(2) Burning

Exhaust rate and composition of fuel directly influences tritium inventory. The range of anticipated exhausted rate and composition vary according to the plasma operating cycle. Thus, the following items in the simulation in the burning phase must be evaluated ;

- exhaust rate to vacuum pump,
- plasma edge composition, density, pressure, temperature,
- dependence of D-T burn-up fraction on the parameters such as edge temperature, edge composition and density, density profile of D, T, P, and He pumping.

The anticipated exhaust rate would be an exciting burden to trace the controllability on the time response in the downstream components connecting process and process controllers. The simple burning model is adopted to determine the exhaust rate to vacuum pump. Burning speed of D-T fuel, $Burn(t)$, depending on the nominal fusion power could be formulated by the following expressions;

$$Burn(t) = f_{power} \cdot \kappa_p P_{fusion} \cdot \frac{1}{\kappa_E E_f \cdot N_0} (\frac{mole}{h})$$
$$= 2.12 f_{power} (\frac{mole}{GW \cdot h}) \tag{2.4}$$

where,

f_{power}: normalization factor of simulated pulse operation,

P_{fusion}: related power (1000MW),

E_f: D,T fusion energy (17.5MeV),

κ_p, κ_E:conversion factor,

N_0: Avogadro's Number.

Fig. 2.4 shows the simulation result of burning model established by the equation (2.4) extended over 36 pulse operation.

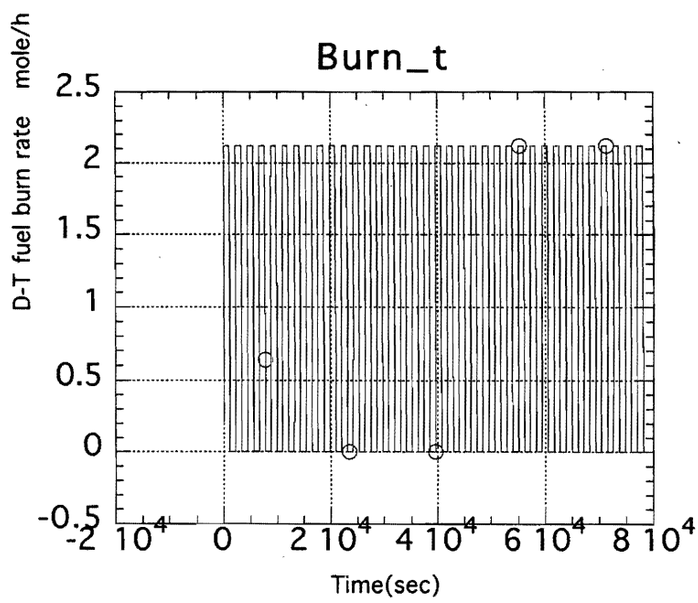


Fig. 2.4 Simulated burn cycle

(3) Fueling and Exhaust rate in Burning

Fueling and exhaust rates of the fuel must be evaluated in the following period of time to extend the burning time while nominal pulse operation scenario is lasting 2 days. The fueling and exhaust rates are interdependent in the requirements to extend the burning time, so that the amount of fuel particles fueled through several devices must be controlled for each operating campaign. The accounts of fueling to extend the burning time could be expressed by the following expressions;

$$Fuel_{f_t}(t) = Fuel_{_t}(t) - F_{plas_nb_t}(t) - F_{plas_prop_t}(t) \tag{2.5}$$

$$Fuel_{f_t}(t) = \frac{Burn_{_t}(t)}{\varepsilon_{bu}} \tag{2.6}$$

$$Fuel_{_d}(t) = F_{plas_in_d}(t) + F_{gas_d}(t) + F_{plas_nb_d}(t) + F_{plas_prop_d}(t) \tag{2.7}$$

$$Fuel_{_p}(t) = F_{plas_in_p}(t) + \chi_{gas_p} \cdot F_{gas_tot}(t) + F_{plas_prop_p}(t) \tag{2.8}$$

where,

$Fuel_{_t}(t)$: total tritium fueled to the plasma (mole/h)

$Fuel_{f_t}(t)$: amount of tritium deliberately fueled to the plasma (mole/h)

$F_{plas_nb_t}(t)$: tritium flow to plasma from neutral beam injection (mole/h)

$F_{plas_prop_t}(t)$: tritium injected to the plasma from pellet injector (mole/h)

ε_{bu}: burn-up ratio (0.05)

$Fuel_{_d}(t)$: total deuterium fueled to the plasma (mole/h)

$F_{plas_in_d}(t)$: deuterium flow to the plasma from pellet injector (mole/h)

$F_{gas_d}(t)$: deuterium flow to the plasma from gas puffer (mole/h)

$F_{plas_nb_d}(t)$: deuterium flow to the plasma from neutral beam injection (mole/h)

$F_{plas_prop_d}(t)$: deuterium injected to the plasma from pellet injector (mole/h)

$Fuel_{_p}(t)$: total protium sources to the plasma (mole/h)

$F_{plas_in_p}(t)$: protium injected to the plasma from pellet injector (mole/h)

$F_{gas_tot}(t)$: total flow to plasma from gas puffer (mole/h)

$F_{plas_prop_p}(t)$: protium injected to plasma from pellet injector propellant (mole/h)

χ_{gas_p}: mole fraction of protium in gas puffer stream.

Amount of tritium deliberately fueled $Fuel_{f_t}(t)$ must be supplemented to meet the following relations;

$$Fuel_{f_t}(t) = F_{plas_in_t}(t) + F_{gas_t}(t) \tag{2.9}$$

where,

$F_{plas_in_t}(t)$: tritium flow to the plasma from pellet injector (mole/h)

$F_{gas_t}(t)$: tritium flow to the plasma from gas puffer (mole/h).

The plasma chamber exhaust flows would be expressed by the following

expressions;

$$F_{ex_t}(t) = Fuel_{_t}(t)(1-\varepsilon_{bu}) - \frac{\Delta PFC_{_t}(t)}{\Delta t} \tag{2.10}$$

$$F_{ex_d}(t) = Fuel_{_d}(t) - Burn_{_d}(t) - \frac{\Delta PFC_{_d}(t)}{\Delta t} \tag{2.11}$$

$$F_{ex_p}(t) = Fuel_{_p}(t) + Gen_{_p}(t) - \frac{\Delta PFC_{_p}(t)}{\Delta t} \tag{2.12}$$

$$F_{ex_he}(t) = Burn_{_t}(t) \tag{2.13}$$

$$Burn_{_d}(t) = Burn_{_t}(t) \tag{2.14}$$

$$Gen_{_p}(t) = \varepsilon_{prot} \cdot \varepsilon_{bu} \cdot Burn_{_t}(t) \tag{2.15}$$

where,

$F_{ex_t}(t)$: exhaust flow of tritium from plasma (mole/h)

$F_{ex_d}(t)$: exhaust flow of deuterium from plasma (mole/h)

$F_{ex_p}(t)$: exhaust flow of protium from plasma (mole/h)

$Burn_{_d}(t)$: deuterium burned in the plasma (mole/h)

$Gen_{_p}(t)$: formation rate protium from fuel reactions (mole/h)

ε_{prot} :relative reaction rate of proton (0.01).

The term, $\Delta PFC_{_k}(t)\ k = t,d,p$, in the equation (2.10), (2.11), (2.12) is introduced to compete the material balance in the burning. Particle exchange to the plasma facing components in burning is accounted for particle losses including bulk intake, codeposited, and erosion. The first one of the established model, bulk intake, was cited from reference [5]. The latter ones, design values of codeposited and erosion, were derived from reference [1]. The particle exchange to the plasma facing components could be expressed by the following expressions;

$$PFC_{_k}(t) = Bulk_{_k}(t) + Co_d_{_k}(t) + Dust_{_k}(t)\ (k = t,d,p) \tag{2.16}$$

$$Bulk_{PFC_k}(t) = \frac{Bulk(T,t) \cdot V_{PFC}}{3.0} \cdot \chi_{edge_k}(t-\Delta t)\ \ (PFC = div, fw) \tag{2.17}$$

$$Bulk_{_k}(t) = Bulk_{div_k}(t) + Bulk_{fw_k}(t) \tag{2.18}$$

$$\chi_{edge_k}(t) = \frac{F_{ex_k}(t)}{F_{ex_t}(t) + F_{ex_d}(t) + F_{ex_p}(t)}\ (k = t,d,p) \tag{2.19}$$

where,

$Bulk_{_k}(t)$: bulk intake of k particle (moles)

$Co_d_{_k}(t)$: codeposited onto plasma facing components of k particle (moles)

$Dust_{_k}(t)$: erosion product of k particle (moles)

$Bulk(T,t)$: inventory concentration of tritium (g/cm3)

V_{PFC} : total volume of divertor/firstwall (4.0/22.4m3).

On each simulation step, Δt, differential equation is established as follows;

$$\frac{\Delta Bulk_{_t}(t_i)}{\Delta t} = \frac{Bulk_{_t}(t_i) - Bulk_{_t}(t_{i-1})}{\Delta t} - \lambda_t \cdot Bulk_{_t}(t_i) \tag{2.20}$$

$$\frac{\Delta Bulk_{_k}(t_i)}{\Delta t} = \frac{Bulk_{_k}(t_i) - Bulk_{_k}(t_{i-1})}{\Delta t} \quad (k = d, p)$$

$$Bulk_{_k}(t) = Bulk_{_t}(t) \cdot \frac{\chi_{edge_k}(t)}{\chi_{edge_t}(t)} \tag{2.21}$$

$$\frac{\Delta Co_d_{_t}(t)}{\Delta t} = \frac{1000}{8760} \cdot \frac{Co_d(M,T)}{3.0} \cdot \chi_{edge_t}(t - \Delta t) - \lambda_t \cdot Co_d_{_t}(t) \tag{2.22}$$

$$\frac{\Delta Co_d_{_k}(t)}{\Delta t} = \frac{1000}{8760} \cdot \frac{Co_d(M,T)}{3.0} \cdot \chi_{edge_k}(t - \Delta t) \quad (k = d, p) \tag{2.23}$$

$$\frac{\Delta Dust_{_t}(t)}{\Delta t} = \frac{1}{8760} \cdot \frac{Dust(M,T) \cdot A_d \cdot dc(M) \cdot \rho(M)}{3.0} \cdot \chi_{edge_t}(t - \Delta t) - \lambda_t \cdot Dust_{_t}(t) \tag{2.24}$$

$$\frac{\Delta Dust_{_k}(t)}{\Delta t} = \frac{1}{8760} \cdot \frac{Dust(M,T) \cdot A_d \cdot dc(M) \cdot \rho(M)}{3.0} \cdot \chi_{edge_k}(t - \Delta t) \quad (k = d, p) \tag{2.25}$$

where,

λ_t : disintegration constant of tritium (6.38E-6h)

A_d : divertor surface area (2.0E2m2)

$dc(M)$: dust concentration ratio (1.0E-1kgtritium/kgdust)

$\rho(M)$: density of material M (kg/m3).

Simulation results of fueling about each particle are described in the Fig. 2.5, Fig. 2.6, Fig. 2.7, respectively. Simulation results of exhaust about each particle are described in the Fig. 2.8, Fig. 2.9, Fig. 2.10, Fig. 2.11, respectively. Time changes of tritium losses including bulk intake, codeposited and erosion in burning are described in the Fig. 2.12, Fig. 2.13, Fig. 2.14, Fig. 2.15, respectively.

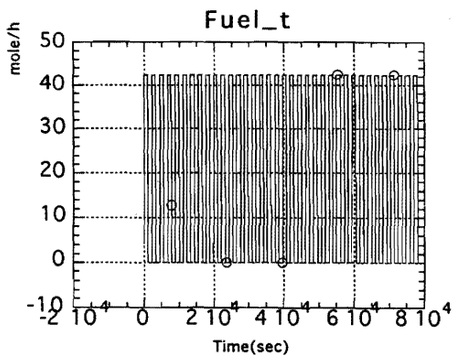


Fig. 2.5 Total tritium fueled to the plasma (mole/h) : Fuel_t(t)

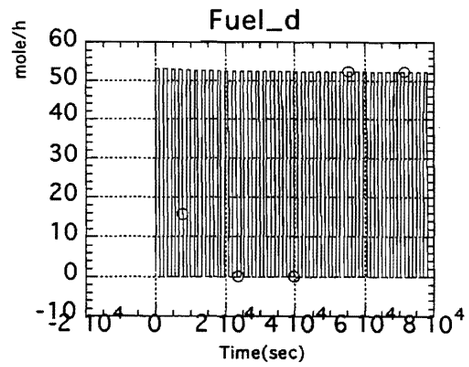


Fig. 2.6 Total deuterium fueled to the plasma (mole/h) : Fuel_d(t)

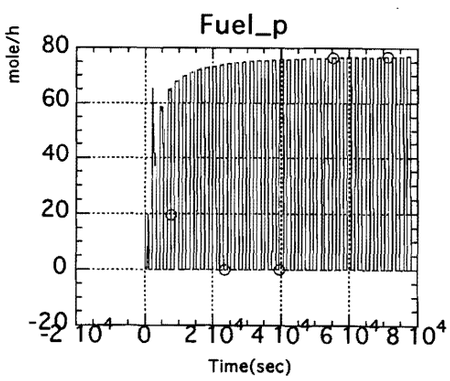


Fig. 2.7 Total protium sources to the plasma (mole/h) : Fuel_p(t)

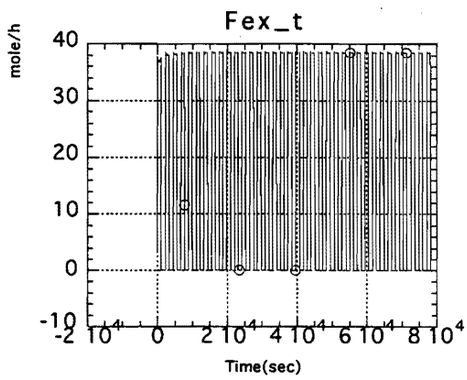


Fig. 2.8 Exhaust flow of tritium from plasma (mole/h) : Fex_t(t)

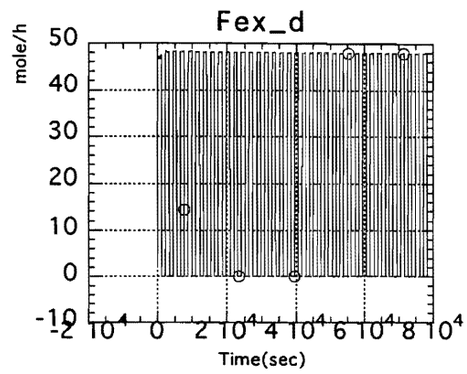


Fig. 2.9 Exhaust flow of deuterium from plasma (mole/h) : Fex_d(t)

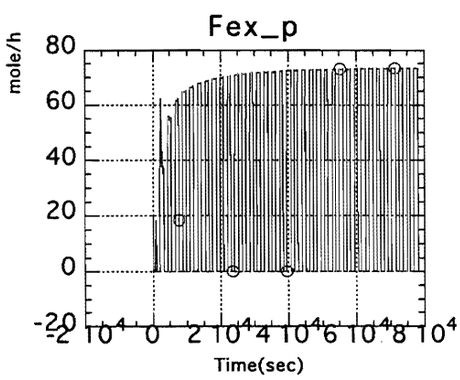


Fig. 2.10 Exhaust flow of protium from plasma (mole/h) : Fex_p(t)

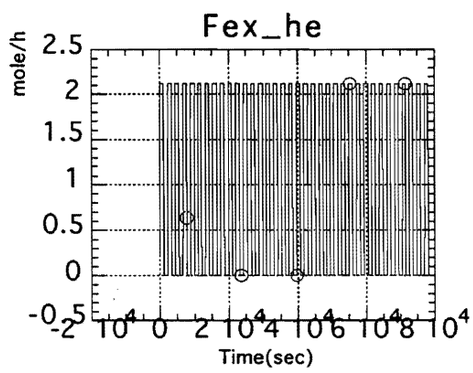


Fig. 2.11 Exhaust flow of helium from plasma (mole/h) : Fex_he(t)

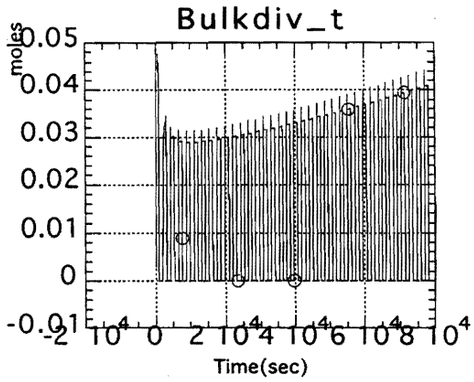


Fig. 2.12 Bulk intake of tritium from the edge into divertor (moles): Bulkdiv_t(t)

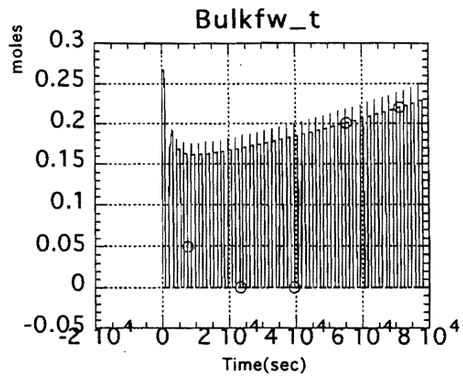


Fig. 2.13 Bulk intake of tritium from the edge into first wall (moles) : Bulkfw_t(t)

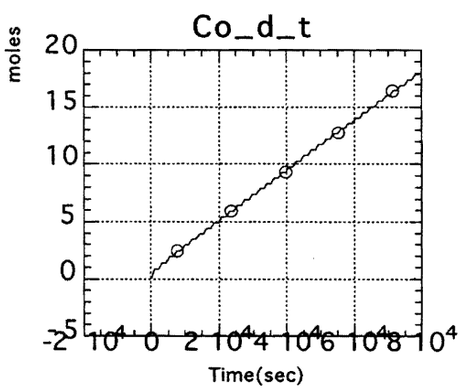


Fig. 2.14 Tritium codeposited onto plasma facing components (moles) : Co_d_t(t)

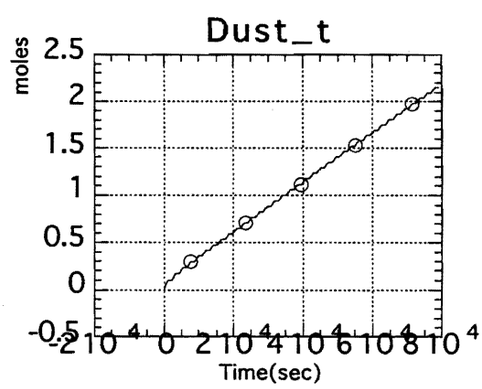


Fig. 2.15 Tritium in erosion product (moles) : Dust_t(t)

3.1.2 Vacuum Pump Subsystem

(1) The control equation

Citing the equation (2.1) called the control equation, let components of the state vector, $\mathbf{x}(t)$, in the equation (2.1) be Tvacuum(t), Dvacuum(t), Pvacuum(t) and Hevacuum(t) denoting the variance in the time of the inventory of each particle in exhaust processing stream. The matrix **A** in the equation (2.1) represents the system matrix employed in the cryopump

mechanical scraping rate. The excitation vector components which are denoting a content of the each particle in the exhaust stream are derived from the equation (2.10), (2.11), (2.12), and (2.13). Thus, control equation in the vacuum pump subsystem could be expressed by the following expressions;

$$\frac{d}{dt}\begin{bmatrix} T_{vacuum}(t) \\ D_{vacuum}(t) \\ P_{vacuum}(t) \\ He_{vacuum}(t) \end{bmatrix} = -\frac{\varepsilon_{scrap}\cdot r_{scrap}}{a_{cryo}}\begin{bmatrix} T_{vacuum}(t) \\ D_{vacuum}(t) \\ P_{vacuum}(t) \\ He_{vacuum}(t) \end{bmatrix} + \frac{1}{n_{pump}}\begin{bmatrix} F_{ex_t}(t) \\ F_{ex_d}(t) \\ F_{ex_p}(t) \\ F_{ex_he}(t) \end{bmatrix} \tag{2.26}$$

where,

$T_{vacuum}(t)$: tritium inventory per pump (moles)

$D_{vacuum}(t)$: deuterium inventory per pump (moles)

$P_{vacuum}(t)$: protium inventory per pump (moles)

$He_{vacuum}(t)$: helium inventory per pump (moles)

ε_{scrap} : removal efficiency of cryodeposit (0.2)

r_{scrap} : scraping rate on cryosurface (1.92m2/h)

a_{cryo} : cryosurface per pump (0.032m2)

n_{pump} : number of operating pumps (16).

The exhaust processing stream is performed by using the cryopump adsorption. The cryopump system consists of 24 pumps, 16 pumping while the remaining 8 are regenerating. Fig. 2.16 shows the simulating cryopump cycle description during partial regeneration.

(2) The observation equation

The equation (2.2) is called the observation equation. The observation matrix in the equation (2.2), **C**, must be defined by the functions of the cryopump mechanical scraping rate. Thus, observation equation in the vacuum pump subsystem could be expressed by the following expressions;

$$\mathbf{y} = \frac{\varepsilon_{scrap}\cdot r_{scrap}}{a_{cryo}}\cdot\begin{bmatrix} 1&0&0&0 \\ 0&1&0&0 \\ 0&0&1&0 \\ 0&0&0&1 \end{bmatrix}\begin{bmatrix} T_{vacuum}(t) \\ D_{vacuum}(t) \\ P_{vacuum}(t) \\ He_{vacuum}(t) \end{bmatrix} \tag{2.27}$$

$$= \begin{bmatrix} R_{vacuum_t}(t) \\ R_{vacuum_d}(t) \\ R_{vacuum_p}(t) \\ R_{vacuum_he}(t) \end{bmatrix} \tag{2.28}$$

$$F_{v_out}(t) = (R_{vacuum_t}(t) + R_{vacuum_d}(t) + R_{vacuum_p}(t) + R_{vacuum_he}(t))\cdot n_{pump} \tag{2.29}$$

where,

$F_{v_out}(t)$: total flow from cryopumps to the fuel clean-up subsystem (mole/h).

The variance in the time of the state and the observation vector components defined by the equation (2.26) and (2.28) is simulated for the following period of time while the pulse operation cycle is lasting 2 days. Fig. 2.17 shows the flow diagram of the exhaust processing in the vacuum pump subsystem with the spatial distributions of the vector components whose values must be evaluated. The simulation results of the state vector components are described in the Fig. 2.18, Fig. 2.19, Fig. 2.20, Fig. 2.21, respectively. The simulation results of the observation vector components and $F_{v_out}(t)$ defined by the equation (2.29) are described in the Fig. 2.22, Fig. 2.23, Fig. 2.24, Fig. 2.25, and Fig. 2.26.

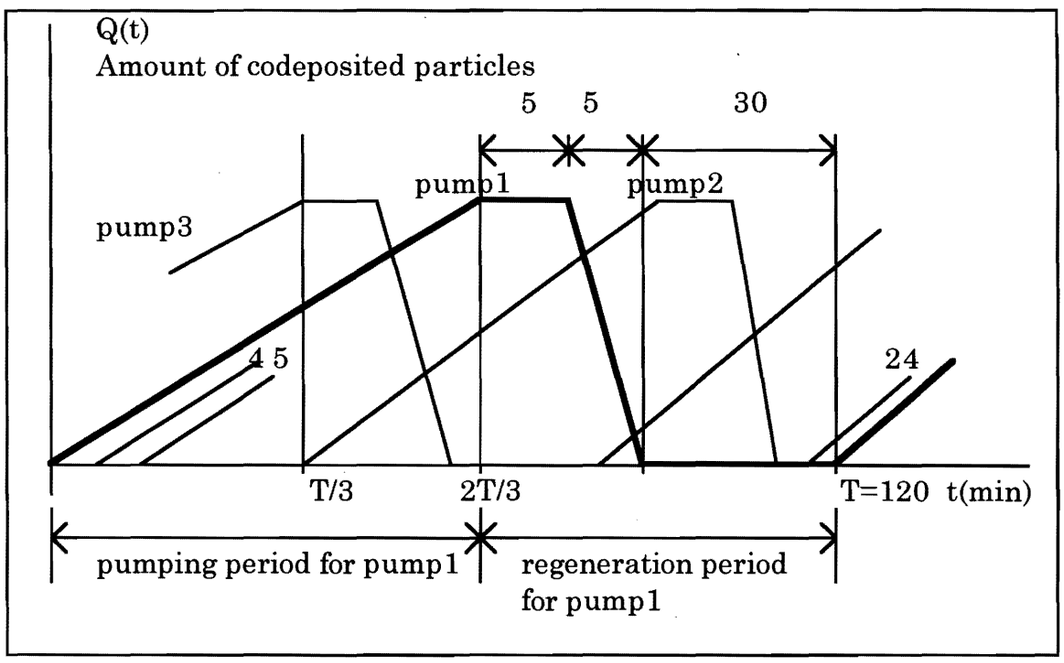


Fig. 2.16 The simulated cryopump cycle

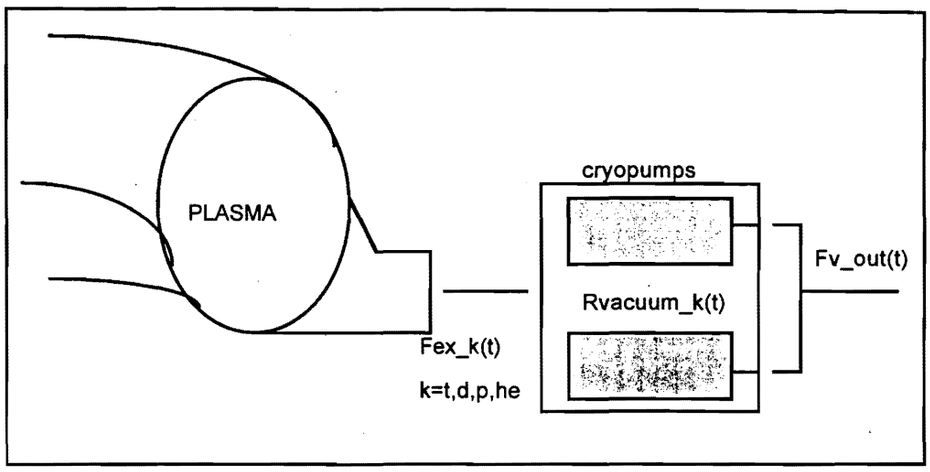


Fig. 2.17 Flow diagram of the exhaust processing

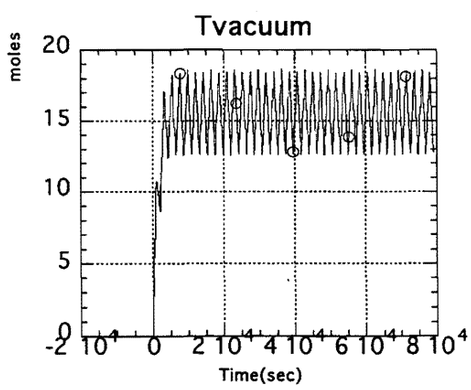


Fig. 2.18 Tritium inventoty on the cryopumps (moles) : Tvacuum(t)

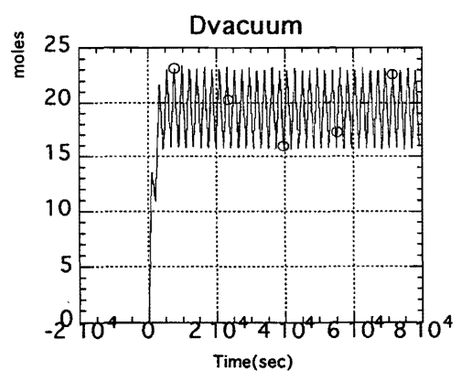


Fig. 2.19 Deuterium inventory on the cryopumps (moles) : Dvacuum(t)

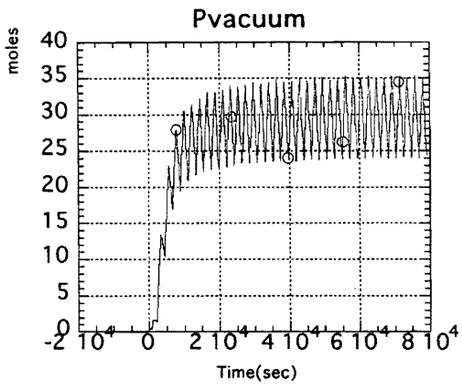


Fig. 2.20 Protium inventory on the cryopumps (moles) : Pvacuum(t)

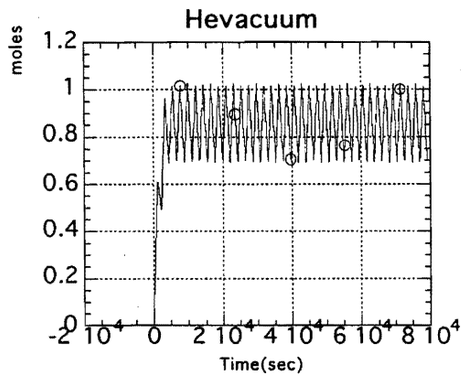


Fig. 2.21 Herium inventory on the cryopumps (moles) : Hevacuum(t)

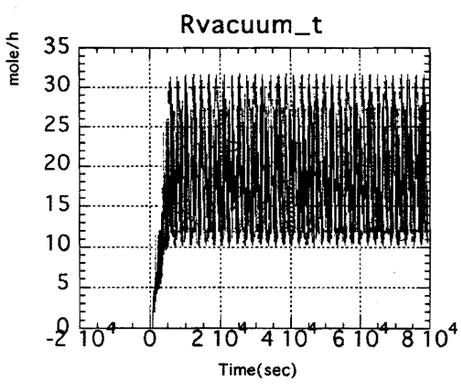


Fig. 2.22 Removal rate of tritium from continuos regeneration cryopump (mole/h) : Rvacuum_t(t)

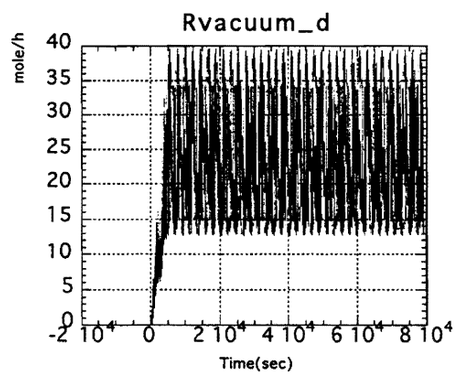


Fig. 2.23 Removal rate of deuterium from continuos regeneration cryopump (mole/h) : Rvacuum_d(t)

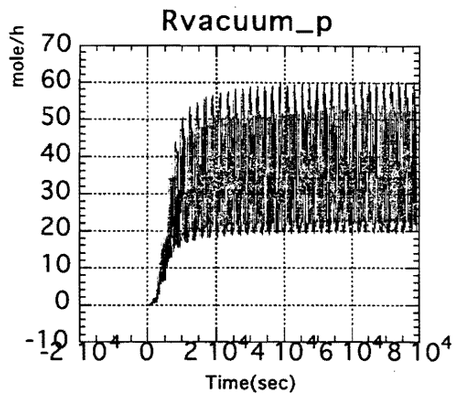


Fig. 2.24 Removal rate of protium from continuous regeneration cryopump (mole/h) : Rvacuum_p(t)

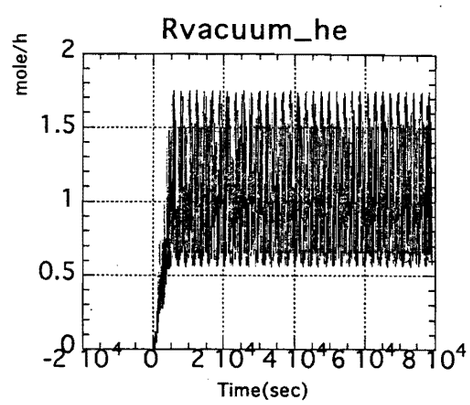


Fig. 2.25 Removal rate of herium from contenuous regeneration cryopump (mole/h) : Rvacuum_he(t)

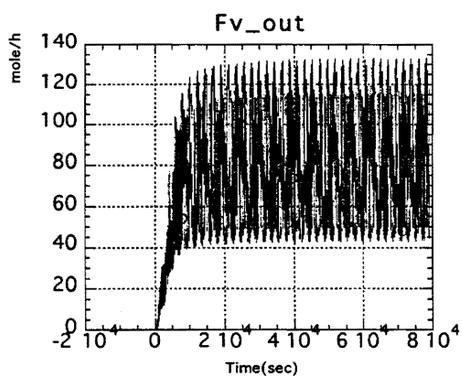


Fig. 2.26 Total flow from cyopumps to the fuel clean-up subsystem (mole/h) : Fv_out(t)

3.1.3 Fuel Clean-up Subsystem

The space vector components are assigned to the content of each particle in the fuel clean-up processing stream. The variance in the time of the space vector components could be followed by observing the component responses which are reflected by the each unit function in the fuel clean-up subsystem. The fuel clean-up subsystem consists of molecular sieve beds

and palladium membrane reactor. The function of the fuel clean-up, purifying, standby, and regenerating is formulated by using the model composed of the three sets of molecular sieve beds. According to the progress in the fuel clean-up processing, one set would be purifying the exhaust flow from plasma chamber, one set would be in standby, and the other set would be regenerating. Fig. 2.27 shows the flow diagram of the fuel clean-up subsystem.

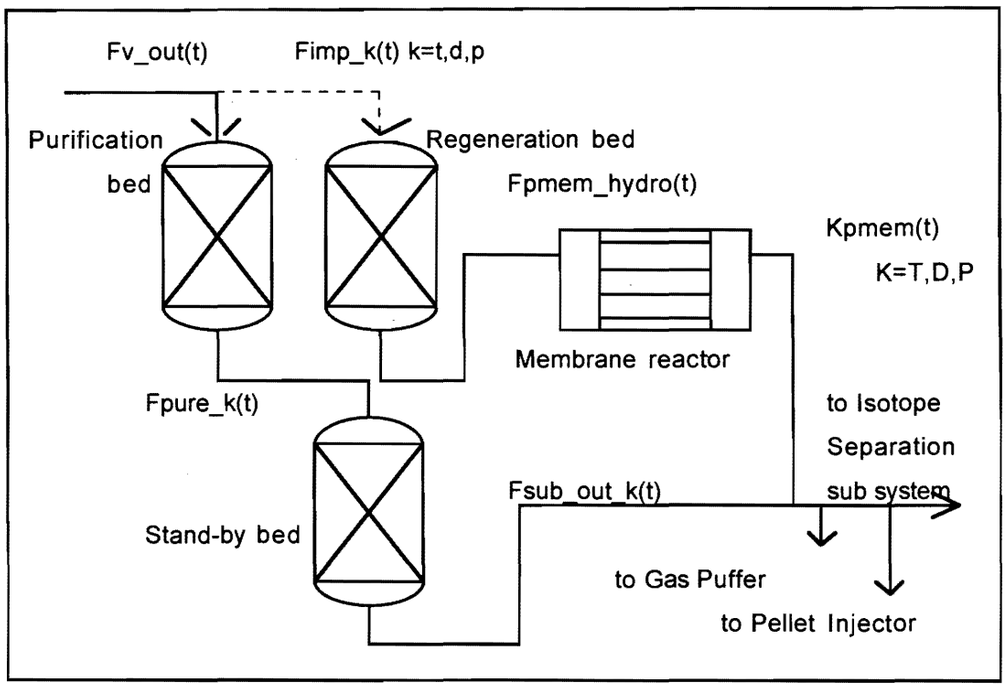


Fig. 2.27 Flow diagram of the fuel clean-up subsystem

3.1.3.1 Purification

(1) The control equation

The content of each particle in the exhaust stream was formulated to express the function of a purifying hydrogen including impurities. The space vector components were set to the pure hydrogen formations and the impurities. The flow of each particle into the purification could be expressed by the following expressions;

$$F_{pure_}(t) = F_{v_out}(t) \cdot \omega_{pure_} \tag{2.30}$$

$$F_{imp_}(t) = F_{v_out}(t) \cdot \omega_{imp_} \tag{2.31}$$

$$F_{pure_}(t) = F_{t_pure}(t) + F_{d_pure}(t) + F_{p_pure}(t) + F_{he_}(t) \tag{2.32}$$

$$F_{imp_}(t) = F_{cq4_imp}(t) + F_{nq3_imp}(t) + F_{q2o_imp}(t) + F_{nhydro_imp}(t) \tag{2.33}$$

$$\begin{bmatrix} F_{cq4_imp}(t) \\ F_{uq3_imp}(t) \\ F_{q2o_imp}(t) \end{bmatrix} = F_{imp_}(t) \cdot \begin{bmatrix} \chi_{cq4_} \\ \chi_{nq3_} \\ \chi_{q2o_} \end{bmatrix} \tag{2.34}$$

$$\begin{bmatrix} F_{t_pure}(t) \\ F_{d_pure}(t) \\ F_{p_pure}(t) \end{bmatrix} = n_{pump} \cdot \begin{bmatrix} R_{vacuum_t}(t) \\ R_{vacuum_d}(t) \\ R_{vacuum_p}(t) \end{bmatrix} - F_{hydro_imp_w}(t) \cdot \begin{bmatrix} i_{t_}(t) \\ i_{d_}(t) \\ i_{p_}(t) \end{bmatrix} \tag{2.35}$$

$$F_{hydro_imp_w}(t) = \omega_1 \cdot F_{cq4_imp}(t) + \omega_2 \cdot F_{nq3_imp}(t) + \omega_3 \cdot F_{q2o_imp}(t) \tag{2.36}$$

$$F_{nhydro_imp}(t) = (1 - \chi_{h_imp}) \cdot F_{imp_}(t) \tag{2.37}$$

$$\chi_{h_imp}(t) = \chi_{cq4_} + \chi_{nq3_} + \chi_{q2o_} \tag{2.38}$$

$$\begin{bmatrix} i_{t_}(t) \\ i_{d_}(t) \\ i_{p_}(t) \end{bmatrix} = \frac{1}{R_{vacuum_t}(t) + R_{vacuum_d}(t) + R_{vacuum_p}(t)} \cdot \begin{bmatrix} R_{vacuum_t}(t) \\ R_{vacuum_d}(t) \\ R_{vacuum_p}(t) \end{bmatrix} \tag{2.39}$$

where,

$F_{pure_}(t)$: flow of pure hydrogen to the purification (mole/h)

$F_{imp_}(t)$: flow of hydrogen containing impurities into the fuel clean-up (mole/h)

$\omega_{pure_}$: generic hydrogen (T,D,P) and He mole fraction in the plasma chamber exhaust (0.98)

$\omega_{imp_}$: impurities fraction in the plasma chamber exhaust (0.02)

$F_{t_pure}(t)$: flow of pure tritium to the purification (mole/h)

$F_{d_pure}(t)$: flow of pure deuterium to the purification (mole/h)

$F_{p_pure}(t)$: flow of pure protium to the purification (mole/h)

$F_{he_}(t)$: flow of helium to the purification (mole/h)

$F_{cq4_imp}(t)$: flow of methane to the purification (mole/h)

$F_{nq3_imp}(t)$: flow of ammonia to the purification (mole/h)

$F_{q2o_imp}(t)$: flow of water to the purification (mole/h)

$F_{nhydro_imp}(t)$: flow of non-hydrogen containing impurities to the purification (mole/h)

$\chi_{cq4_}$: specified mole fraction of methane in flow to the purification (5.6E-1)

$\chi_{nq3_}$: specified mole fraction of ammonia in flow to the purification (4.0E-2)

$\chi_{q2o_}$: specified mole fraction of water in flow to the purification (8.0E-2)

$F_{hydro_imp_w}(t)$: hydrogen flow containing impurities (mole/h)

$i_{t_}(t)$: isotopic fraction of tritium on the vacuum pump subsystem

$i_{d_}(t)$: isotopic fraction of deuterium on the vacuum pump subsystem

$i_{p_}(t)$: isotopic fraction of protium on the vacuum pump subsystem

ω_1 : mole number (4)

ω_2 : mole number (3)

ω_3 : mole number (2)

χ_{h_imp} : fraction of hydrogen containing impurities in flow to the purification.

In purifying, hydrogen containing impurities has been build up. The variances in the time of the build up hydrogen could be expressed by the following expressions ;

$$\frac{d}{dt}\begin{bmatrix} N_{cq4_imp}(t) \\ N_{nq3_imp}(t) \\ N_{q2o_imp}(t) \end{bmatrix} = F_{imp_}(t)\cdot\begin{bmatrix} \chi_{cq4_} \\ \chi_{nq3_} \\ \chi_{q2o_} \end{bmatrix} \tag{2.40}$$

where,

$N_{cq4_imp}(t)$: methane on the purification (moles)

$N_{nq3_imp}(t)$: ammonia on the purification (moles)

$N_{q2o_imp}(t)$: water on the purification (moles).

The simulation results of the space vector components defined by the equation (2.34) and (2.35) are described in the Fig. 2.28, Fig. 2.29, Fig. 2.30, Fig. 2.31, Fig. 2.32, and Fig. 2.33, respectively.

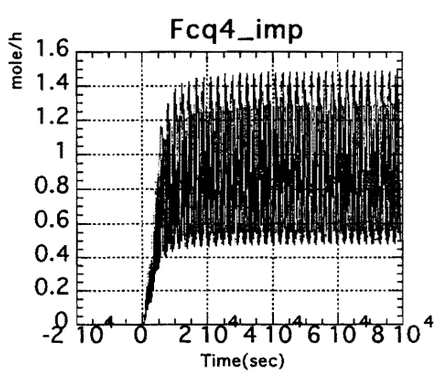


Fig. 2.28 Flow of methane to the purification (mole/h) : Fcq4_imp(t)

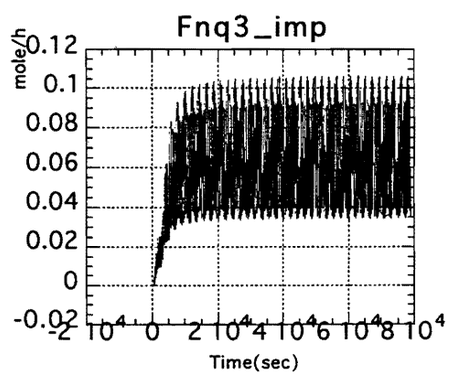


Fig. 2.29 Flow of ammonia to the purification (mole/h) : Fnq3_imp(t)

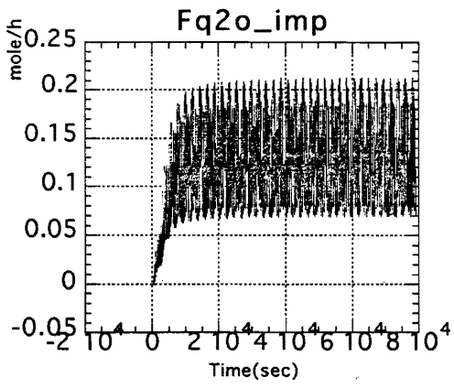


Fig. 2.30 Flow of water to the purification (mole/h) : Fq2o_imp(t)

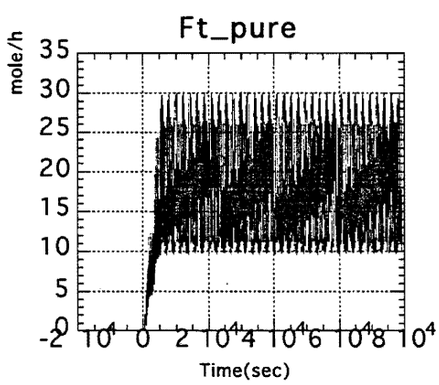


Fig. 2.31 Flow of pure tritium to the purification (mole/h) : Ft_pure(t)

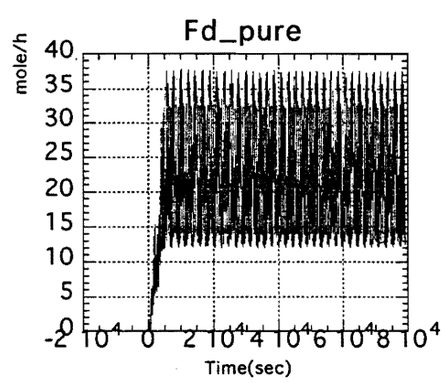


Fig. 2.32 Flow of pure deuterium to the purification (mole/h) : Fd_pure(t)

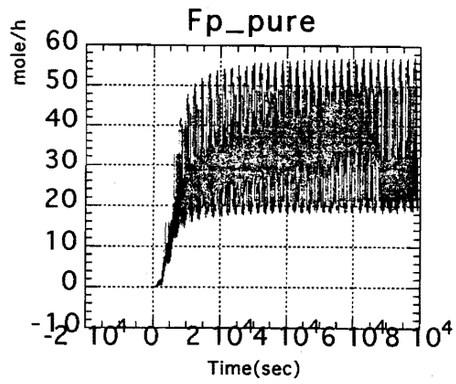


Fig. 2.33 Flow of pure protium to the purification (mole/h) : Fp_pure(t)

3.1.3.2 Standby

(1) The observation equation

The one bed to be used for the purification in the next step according to the subsequent procession is preloaded with hydrogen. The preloading is exploited by using the stream that leaves the purification bed containing pure hydrogen. The control equation in the standby could be expressed by the following expressions;

$$\begin{bmatrix} F_{t2_sub_in}(t) \\ F_{d2_sub_in}(t) \\ F_{p2_sub_in}(t) \end{bmatrix} = \frac{1}{2} \cdot \begin{bmatrix} F_{t_pure}(t) \\ F_{d_pure}(t) \\ F_{p_pure}(t) \end{bmatrix} + F_{imp_}(t) \cdot \begin{bmatrix} \chi_{t_sub}(t) \\ \chi_{d_sub}(t) \\ \chi_{p_sub}(t) \end{bmatrix} \tag{2.41}$$

$$F_{h2_sub_in}(t) = F_{t2_sub_in}(t) + F_{d2_sub_in}(t) + F_{p2_sub_in}(t) \tag{2.42}$$

$$\frac{dN_s(t)}{dt} = -\frac{N_s(t)}{\tau_{sb}} \tag{2.43}$$

$$\tau_{sb} = \frac{N_{s_\max}}{F_{h2_sub_in}(t)} \tag{2.44}$$

$$\frac{dN_s(t)}{dt} = \frac{-N_s \cdot (F_{t2_sub_in}(t) + F_{d2_sub_in}(t) + F_{p2_sub_in}(t)}{N_{s_\max}} \tag{2.45}$$

$$\begin{bmatrix} \chi_{t_sub}(t) \\ \chi_{d_sub}(t) \\ \chi_{p_sub}(t) \end{bmatrix} = \frac{1}{F_{t2_sub_in}(t) + F_{d2_sub_in}(t) + F_{p2_sub_in}(t)} \cdot \begin{bmatrix} F_{t2_sub_in}(t) \\ F_{d2_sub_in}(t) \\ F_{p2_sub_in}(t) \end{bmatrix} \tag{2.46}$$

where,

$F_{t2_sub_in}(t)$: flow of pure tritium molecules to the standby (mole/h)

$F_{d2_sub_in}(t)$: flow of pure deuterium molecules to the standby (mole/h)

$F_{p2_sub_in}(t)$: flow of pure protium molecules to the standby (mole/h)
$F_{h2_sub_in}(t)$: flow of pure hydrogen molecules to the standby (mole/h)
$N_s(t)$: number of unfilled adsorption sites for hydrogen on the standby bed (mole/h)
τ_{sb} : time constant for preloading the standby bed
$N_{s_\max}$: maximum number of unfilled adsorption sites for hydrogen on the standby bed (17.7mole)
$\chi_{t_sub}(t)$: mole fraction of tritium entering the standby
$\chi_{d_sub}(t)$: mole fraction of deuterium entering the standby
$\chi_{p_sub}(t)$: mole fraction of protium entering the standby.

The simulation results of the observation vector components defined by the equation (2.41) are described in the Fig. 2.34, Fig. 2.35, and Fig. 2.36, respectively.

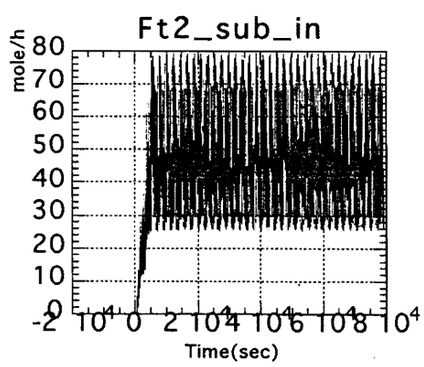


Fig. 2.34 Flow of tritium to the standby (mole/h) : Ft2_sub_in(t)

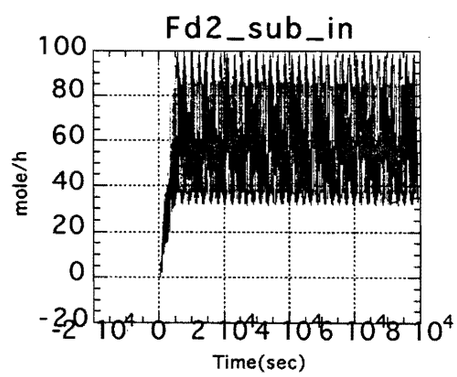


Fig. 2.35 Flow of deuterium to the standby (mole/h) : Fd2_sub_in(t)

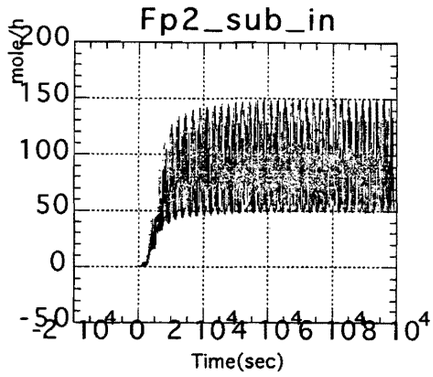


Fig. 2.36 Flow of protium to the standby (mole/h) : Fp2_sub_in(t)

Formulating the observation equation about the outlet of standby, the observation matrix, **C**, in the equation (2.2) is unit vector. The transfer matrix, **D**, in the equation (2.2) could be defined by an amounts of the buildup of number of adsorption sites whose variances are determined by equation (2.43). The observation equation could be expressed by the following expressions;

$$\frac{1}{2}\begin{bmatrix} F_{t_sub_out}(t) \\ F_{d_sub_out}(t) \\ F_{p_sub_out}(t) \end{bmatrix} = \begin{bmatrix} 100 \\ 010 \\ 001 \end{bmatrix} \begin{bmatrix} F_{t2_sub_in}(t) \\ F_{d2_sub_in}(t) \\ F_{p2_sub_in}(t) \end{bmatrix} - \frac{dN_s(t)}{dt} \begin{bmatrix} \chi_{t_sub}(t) \\ \chi_{d_sub}(t) \\ \chi_{p_sub}(t) \end{bmatrix} \tag{2.47}$$

where,

$F_{t_sub_out}(t)$: flow of pure tritium out of the standby (mole/h)

$F_{d_sub_out}(t)$: flow of pure deuterium out of the standby (mole/h)

$F_{p_sub_out}(t)$: flow of pure protium out of the standby (mole/h).

The simulation results of the observation vector components definened by the equation (2.47) are described in the Fig. 2.37, Fig. 2.38, and Fig. 2.39, respectively.

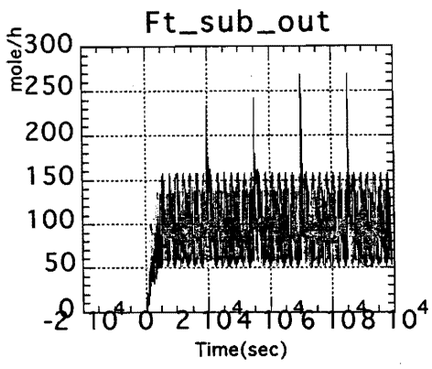


Fig. 2.37 Flow of tritium out of the standby (mole/h) : Ft_sub_out(t)

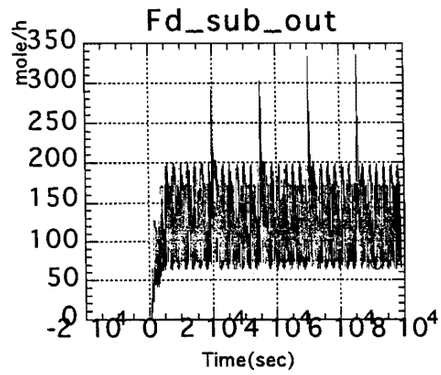


Fig. 2.38 Flow of deuterium out of the standby (mole/h) : Fd-sub_out(t)

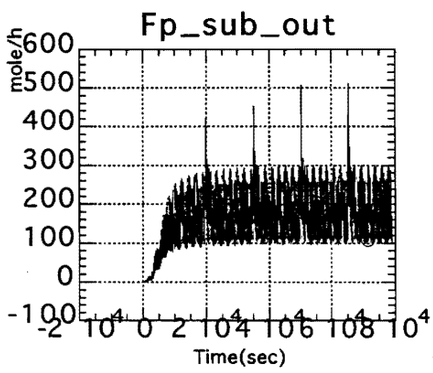


Fig. 2.39 Flow of protium out of the standby (mole/h) : Fp_sub_out(t)

3.1.3.3 Regeneration

(1) The control equation

The evolution of hydrogen containing species in the regeneration process is formulated by using the simple model supposed to be a constant removal rate. Assuming not building up of residual material on the bed in regenerating, the formation of removal rate could be expressed by introducing the regenerating time constant values;

$$E_{cq4_reg}(t) = \frac{N_{cq4_reg}(t)}{t_{msieve_reg}} \tag{2.48}$$

$$E_{nq3_reg}(t) = \frac{N_{nq3_reg}(t)}{t_{msieve_reg}} \tag{2.49}$$

$$E_{q2o_reg}(t) = \frac{N_{q2o_reg}(t)}{t_{msieve_reg}} \tag{2.50}$$

$$\frac{dN_{cq4_reg}(t)}{dt} = F_{imp_cq4}(t) \tag{2.51}$$

$$\frac{dN_{nq3_reg}(t)}{dt} = F_{imp_nq3}(t) \tag{2.52}$$

$$\frac{dN_{q2o_reg}(t)}{dt} = F_{imp_q2o}(t) \tag{2.53}$$

where,

$N_{cq4_reg}(t)$: amount of methane on purification bed at regeneration (moles)

$N_{nq3_reg}(t)$: amount of ammonia on purification bed at regeneration (moles)

$N_{q2o_reg}(t)$: amount of water on purification bed at regeneration (moles)

$E_{cq4_reg}(t)$: evolution of methane from molecular sieve during regeneration (mole/h)

$E_{nq3_reg}(t)$: evolution of ammonia from molecular sieve during regeneration (mole/h)

$E_{q2o_reg}(t)$: evolution of water from molecular sieve during regeneration (mole/h)

t_{msieve_reg} : regeneration time for molecular sieve (4h).

3.1.3.4 Palladium membrane

(1) The control equation

The desorped impurities in the regeneration process are directed to a palladium membrane reactor. In the palladium membrane reactor, freeing pure hydrogen chemically compounded in impurities is accomplished by a pumping themselves that permeate the palladium membrane. The inventories in the palladium membrane reactor could be expressed on the assumption that the permeation time must be constant even through membrane reactor has been in alive. The control equations in the palladium membrane reactor could be expressed by the following expressions;

$$\frac{d}{dt}\begin{bmatrix} T_{pmem}(t) \\ D_{pmem}(t) \\ P_{pmem}(t) \end{bmatrix} = F_{pmem_hydro}(t) \cdot \begin{bmatrix} \chi_{pmem_t}(t) \\ \chi_{pmem_d}(t) \\ \chi_{pmem_p}(t) \end{bmatrix} - \frac{1}{t_{pmem}} \cdot \begin{bmatrix} T_{pmem}(t) \\ D_{pmem}(t) \\ P_{pmem}(t) \end{bmatrix} \tag{2.54}$$

$$F_{pmem_hydro}(t) = 4E_{cq4_reg}(t) + 3E_{nq3_reg}(t) + 2E_{q2o_reg}(t) \tag{2.55}$$

where,

$T_{pmem}(t)$: tritium on palladium membrane reactor (moles)

$D_{pmem}(t)$: deuterium on palladium membrane reactor (moles)

$P_{pmem}(t)$: protium on palladium membrane reactor (moles)

$\chi_{pmem_t}(t)$: isotopic fraction of tritium on molecular sieve at end of purification

$\chi_{pmem_d}(t)$: isotopic fraction of deuterium on molecular sieve at end of purification

$\chi_{pmem_p}(t)$: isotopic fraction of protium on molecular sieve at end of purification

t_{pmem} : time constant of palladium membrane reactor (2h).

The simulation results of the space vector components defined by the equation (2.54) are described in the Fig. 2.40, Fig. 2.41, Fig. 2.42, respectively.

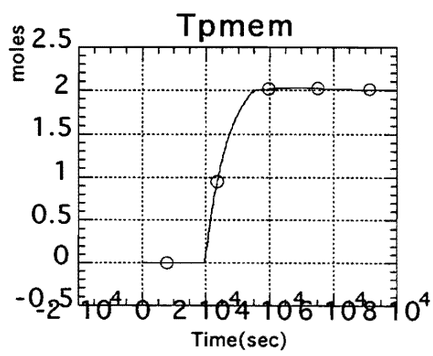


Fig. 2.40 Tritium on palladium membrane reactor (moles) : Tpmem(t)

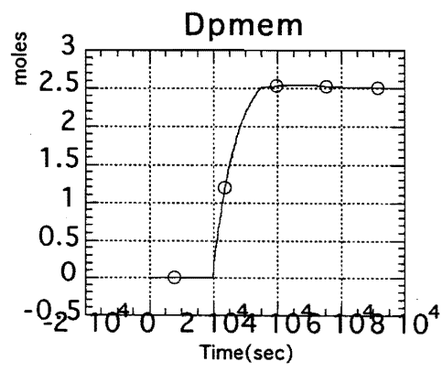


Fig. 2.41 Deuterium on palladium membrane reactor (moles) : Dpmem(t)

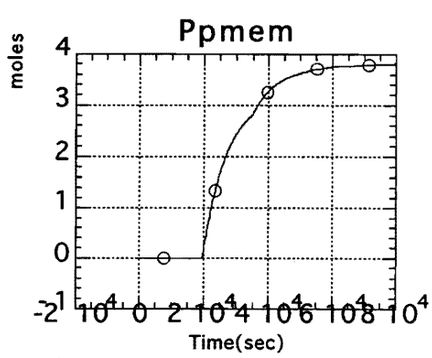


Fig. 2.42 Protium on palladium membrane reactor (moles) : Ppmem(t)

(2) The observation equation

The total flow from the fuel clean-up subsystem to the isotope separation subsystem includes a contribution flowing out of the standby molecular sieve beds and a contribution flowing from the palladium membrane reactor. Then the observation equation in the fuel clean-up subsystem could be expressed by deriving the corresponding terms defined in the equation (2. 47) and (2.54);

$$\begin{bmatrix} F_{t_puri_out}(t) \\ F_{d_puri_out}(t) \\ F_{p_puri_out}(t) \end{bmatrix} = \begin{bmatrix} F_{t_sub_out}(t) \\ F_{d_sub_out}(t) \\ F_{p_sub_out}(t) \end{bmatrix} + \frac{1}{t_{pmem}} \begin{bmatrix} T_{pmem}(t) \\ D_{pmem}(t) \\ P_{pmem}(t) \end{bmatrix} \tag{2.56}$$

where,

$F_{t_puri_out}(t)$: flow of tritium to the isotope separation subsystem (mole/h)

$F_{d_puri_out}(t)$: flow of deuterium to the isotope separation subsystem (mole/h)

$F_{p_puri_out}(t)$: flow of protium to the isotope separation subsystem (mole/h).

The simulation results of the observation vector components defined by the equation (2.56) are described in the Fig. 2.43, Fig. 2.44, and Fig. 2.45, respectively.

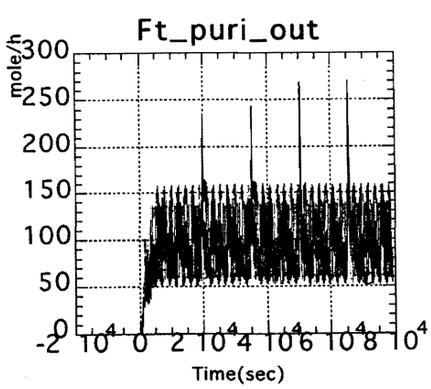


Fig. 2.43 Flow of tritium to the isotope separation subsystem (mole/h) : Ft_pure_out(t)

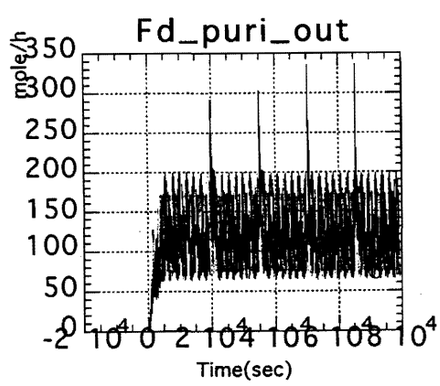


Fig. 2.44 Flow of deuterium to the isotope separation subsystem (mole/h) : Fd_puri_out(t)

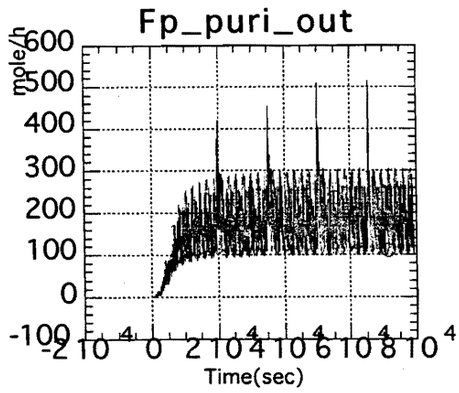


Fig. 2.45 Flow of protium to the isotope separation subsystem (mole/h) : Fp_puri_out(t)

3.1.4 Isotope Separation Subsystem

The isotope separation subsystem has been aimed to make the elementary hydrogen isotopes separate to each other. On the entire fuel processing, the three product stream out of the isotope separation subsystem have been required. These are a tritium rich stream for a fueling to the plasma chamber, a deuterium rich stream for the neutral beam injection and for the fueling, and a protium rich stream for the pellet injector propellant. The cryogenic distillation is put up as the process to accomplish this performance. The performance in the cryogenic distillation is a complex process. On the system design [1], the candidate system consists of a series of cryogenic distillation columns. To trace a mass transfer over a series of columns, establishing the simulation models in details of the separation process is required. Functionally, the isotope separation process is a continuous process. The three products stream out of the isotope separation subsystem would be sensitive in accordance with the time variance in the receiving flow. To provide a time variance of the product stream out of the isotope separation process, a simple simulation model is established. Fig. 2.46 shows the flow diagram of the isotope separation subsystem.

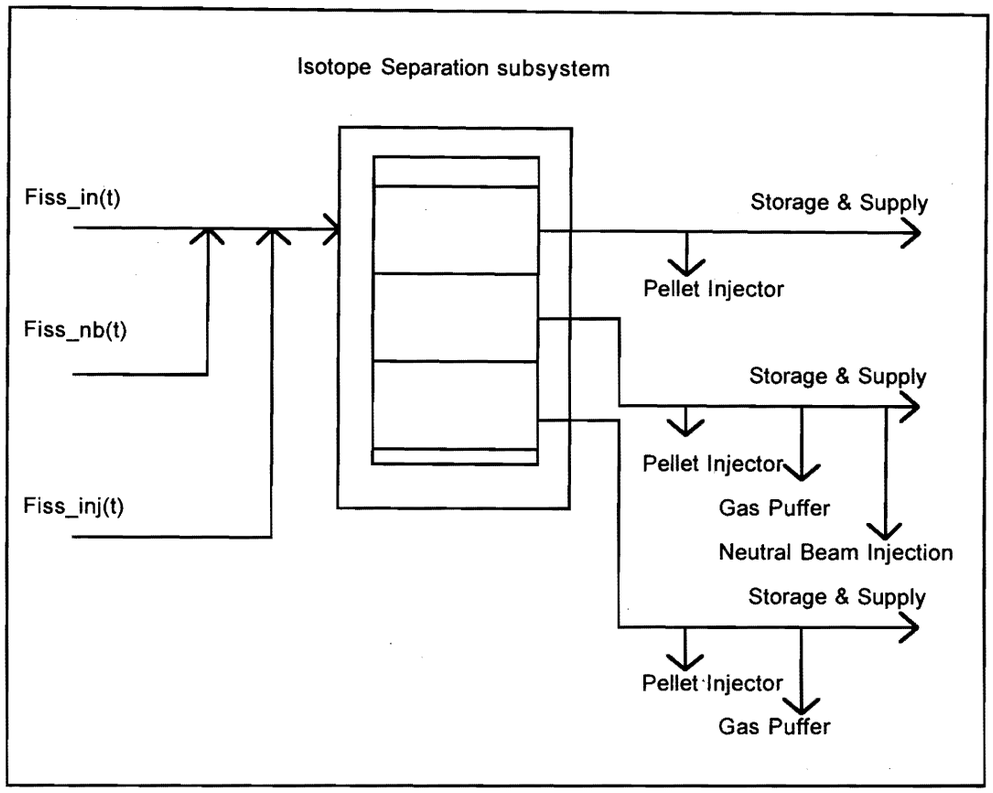


Fig. 2.46 Flow diagram of the isotope separation subsystem

(1) The control equation

Assuming on the condition that product stream out of the isotope separation subsystem depends on the inventory and a system time constant, the simple simulation model was established by the following expressions;

$$\frac{d}{dt}\begin{bmatrix} N_{iss_}(t) \\ T_{iss_}(t) \\ D_{iss_}(t) \end{bmatrix} = \begin{bmatrix} F_{iss_in}(t) \\ F_{iss_in_t}(t) \\ F_{iss_in_d}(t) \end{bmatrix} - \frac{1}{t_{iss}}\begin{bmatrix} N_{iss_}(t) \\ T_{iss_}(t) \\ D_{iss_}(t) \end{bmatrix} \tag{2.57}$$

$$P_{oiss_}(t) = \chi_{iss_p}(t) \cdot N_{iss_}(t) \tag{2.58}$$

$$\begin{bmatrix} F_{iss_in}(t) \\ F_{iss_in_t}(t) \\ F_{iss_in_d}(t) \end{bmatrix} = Frag_{bl} \cdot F_{bl_ccol}(t) \cdot \begin{bmatrix} 1 \\ \chi_{c_t}(t) \\ \chi_{c_d}(t) \end{bmatrix} + Frag_{bl} \cdot F_{bl_}(t) \cdot \begin{bmatrix} 1 \\ \chi_{bl_t}(t) \\ \chi_{bl_d}(t) \end{bmatrix}$$

$$+(1-h_1-h_3) \cdot F_{puri_out}(t) \cdot \begin{bmatrix} 1 \\ \chi_{iss_in_t}(t) \\ \chi_{iss_in_d}(t) \end{bmatrix} \tag{2.59}$$

$$+Frag_{nb} \cdot F_{iss_nb}(t) \cdot \begin{bmatrix} 1 \\ \chi_{nb_t}(t) \\ \chi_{nb_d}(t) \end{bmatrix} + Frag_{iss_inj} \cdot F_{iss_inj}(t) \cdot \begin{bmatrix} 1 \\ \chi_{inj_t}(t) \\ \chi_{inj_d}(t) \end{bmatrix}$$

$$F_{iss_in_p}(t) = F_{iss_in}(t) - F_{iss_in_t}(t) - F_{iss_in_d}(t) \tag{2.60}$$

where,

$N_{iss_}(t)$: total amount of isotopes in the isotope separation subsystem (moles)

$T_{iss_}(t)$: amount of tritium in the isotope separation subsystem (moles)

$D_{iss_}(t)$: amount of deuterium in the isotope separation subsystem (moles)

$P_{oiss_}(t)$: amount of protium in the isotope separation subsystem (moles)

t_{iss} : residence time in the isotope separation subsystem (2.22h)

$F_{iss_in}(t)$: total flow to the isotope separation subsystem (mole/h)

$F_{iss_in_t}(t)$: flow of tritium to the isotope separation subsystem (mole/h)

$F_{iss_in_d}(t)$: flow of deuterium to isotope separation subsystem (mole/h)

$F_{bl_ccol}(t)$: flow from the coolant system (mole/h)

$F_{bl_}(t)$: flow from the blanket system (mole/h)

$F_{puri_out}(t)$: flow from the fuel clean-up subsystem to the isotope separation subsystem (mole/h)

$F_{iss_nb}(t)$: flow from the neutral beam injection to the isotope separation subsystem (mole/h)

$F_{iss_inj}(t)$: flow from the pellet injector to the isotope separation subsystem (mole/h)

$Frag_{bl}$: switch to include the coolant system (0)

$Frag_{nb}$:switch to include the neutral beam injection (1)

$Frag_{iss_inj}$: switch to include the pellet injector (1)

h_1 : fraction of stream from the fuel clean-up subsystem not

processed and directed to the gas puffer (0.3)

h_3 : fraction of stream from the fuel clean-up subsystem not processed and directed to the pellet injector (0.3)

$\chi_{c_t}(t)$: mole fraction of tritium in the coolant system

$\chi_{c_d}(t)$: mole fraction of deuterium in the coolant system

$\chi_{bl_t}(t)$: mole fraction of tritium in the breeder system

$\chi_{bl_d}(t)$: mole fraction of deuterium in the breeder system

$\chi_{iss_in_t}(t)$: mole fraction of tritium in the isotope separation subsystem

$\chi_{iss_in_d}(t)$: mole fraction of deuterium in the isotope separation subsystem

$\chi_{nb_t}(t)$: mole fraction of tritium in stream from the neutral beam injection

$\chi_{nb_d}(t)$: mole fraction of deuterium in stream from the neutral beam injection

$\chi_{inj_t}(t)$: mole fraction of tritium in stream from the pellet injector

$\chi_{inj_d}(t)$: mole fraction of deuterium in stream from the pellet injector.

The simulation results of the space vector components, $T_{iss_}(t)$, $D_{iss_}(t)$, $P_{oiss_}(t)$, defined by the equation (2.57) and (2.58) are described in the Fig. 2.47, Fig. 2.48, Fig. 2.49, respectively. The simulation results of the observation vector components, $F_{iss_in_t}(t)$, $F_{iss_in_d}(t)$, $F_{iss_in_p}(t)$, defined by the equation (2.59) and (2.60) are described in the Fig. 2.50, Fig. 2.51, Fig. 2.52, respectively.

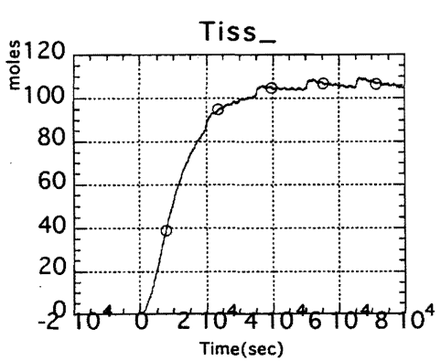


Fig. 2.47 Amount of tritium in the isotope separation subsystem (moles): T_iss(t)

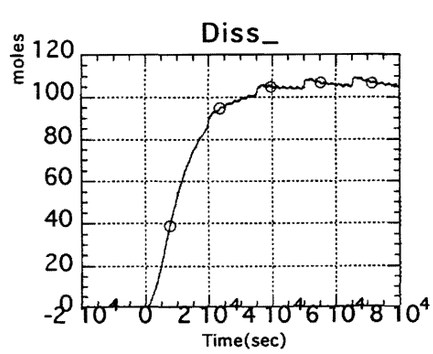


Fig. 2.48 Amount of deuterium in the isotope separation subsystem (moles) : Diss_(t)

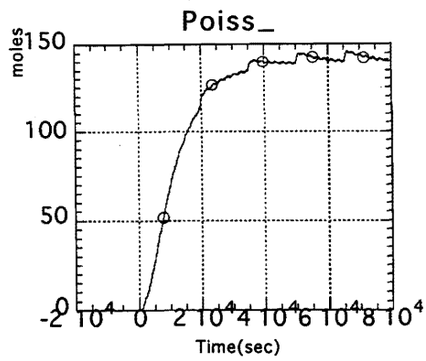


Fig. 2.49 Amount of protium in the isotope separation subsystem (moles) : Poiss(t)

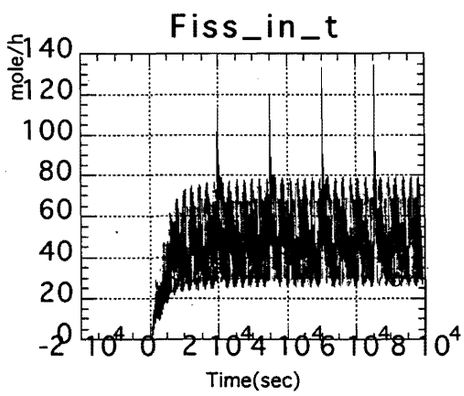


Fig. 2.50 Flow of tritium to the isotope separation subsystem (mole/h) : Fiss_in_t(t)

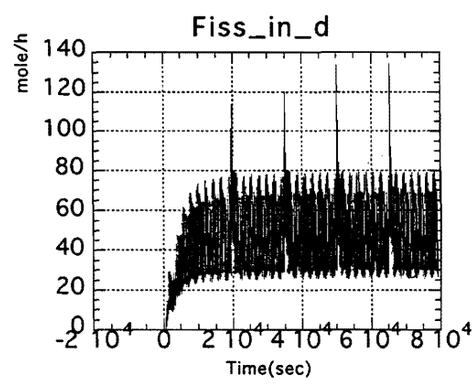


Fig. 2.51 Flow of deuterium to the isotope separation subsystem (mole/h) : Fiss_in_d(t)

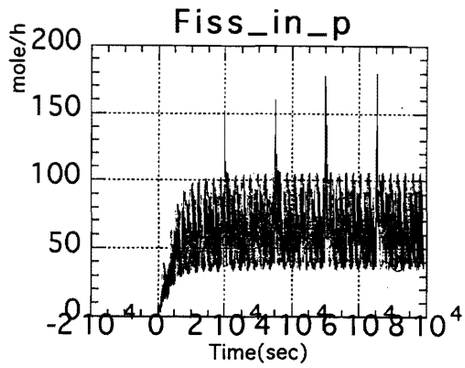


Fig. 2.52 Flow of protium to the isotope separation subsystem (mole/h) : Fiss_in_p(t)

(2) The observation equation

On the assumption that product stream out of the isotope separation subsystem depends on the inventory, the simple simulation model was formulated. The observation vector components in the distillation process could be formulated by considering the equilibrium of the mass transfer over a cryogenic distillation process. Then, observation equation is expressed by the following expressions;

$$F_{iss_top}(t)\cdot\begin{bmatrix}1\\ \chi_{iss_top_t}\\ \chi_{iss_top_t}\end{bmatrix}+F_{iss_mid}(t)\cdot\begin{bmatrix}1\\ \chi_{iss_mid_t}\\ \chi_{iss_mid_d}\end{bmatrix}+F_{iss_bot}(t)\cdot\begin{bmatrix}1\\ \chi_{iss_bot_t}\\ \chi_{iss_bot_d}\end{bmatrix}=\frac{1}{t_{iss}}\begin{bmatrix}N_{iss_}(t)\\ T_{iss_}(t)\\ D_{iss_}(t)\end{bmatrix} \tag{2.61}$$

$$\chi_{iss_top_t}+\chi_{iss_top_d}+\chi_{iss_top_p}=1 \tag{2.62}$$

$$\chi_{iss_mid_t}+\chi_{iss_mid_d}+\chi_{iss_mid_p}=1 \tag{2.63}$$

$$\chi_{iss_bot_t}+\chi_{iss_bot_d}+\chi_{iss_bot_p}=1 \tag{2.64}$$

where,

$F_{iss_top}(t)$: flow from top of the isolation separation subsystem (mole/h)

$F_{iss_mid}(t)$: flow from middle of the isolation separation subsystem (mole/h)

$F_{iss_bot}(t)$: flow from bottom of the isolation separation subsystem

(mole/h)

$\chi_{iss_top_t}$: mole fraction of tritium from top of the isotope separation subsystem

$\chi_{iss_top_t}$: mole fraction of deuterium from top of the isotope separation subsystem

$\chi_{iss_top_p}$: mole fraction of protium from top of the isotope separation subsystem

$\chi_{iss_mid_t}$: mole fraction of tritium from middle of the isotope separation subsystem

$\chi_{iss_mid_d}$: mole fraction of deuterium from middle of the isotope separation subsystem

$\chi_{iss_mid_p}$: mole fraction of protium from middle of the isotope separation subsystem

$\chi_{iss_bot_t}$: mole fraction of tritium from bottom of the isotope separation subsystem

$\chi_{iss_bot_d}$: mole fraction of deuterium from bottom of the isotope separation subsystem

$\chi_{iss_bot_p}$: mole fraction of protium from bottom of the isotope separation subsystem.

The simulation results of the observation vector components, $F_{iss_bot}(t)$, $F_{iss_mid}(t)$, $F_{iss_top}(t)$, defined by the equation (2.61) are described in the Fig.2.53, Fig. 2.54, Fig. 2.55, respectively.

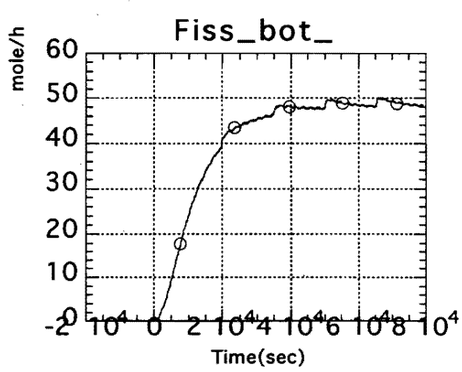


Fig. 2.53 Flow from bottom of the isotope separation subsystem (mole/h) : Fiss_bot_(t)

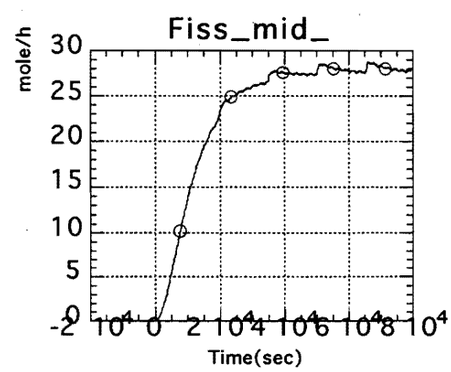


Fig. 2.54 Flow from middle of the isotope separation subsystem (mole/h) : Fiss_mid_(t)

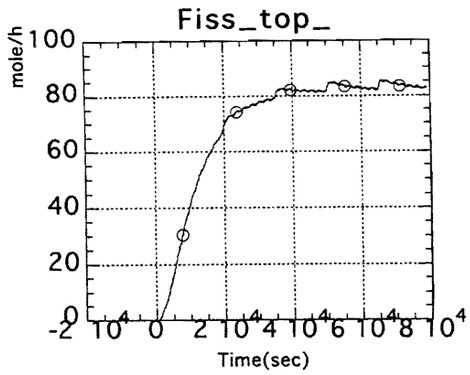


Fig. 2.55 Flow from top of the isotope separation subsystem (mole/h) : Fiss_top_(t)

3.1.5 Neutral Beam Injection Subsystem

The neutral beam has been employed as a supply of the method to be contributed for current drive, current profile control, and plasma heating. On the system design [1], candidate system is based on negative ion beams. The negative ion beams employ deuterium and provide a small contribution for fueling. The neutral beam injection has a large caliber beam port where significant amounts of tritium could be built-up in the back flow from the plasma. The neutral beam is supplemented by a delivery from the deuterium rich stream of the isotope separation subsystem and the deuterium rich stream of the storage and supply subsystem. The flow of the neutral beam must be met to the required flow anticipated to the injection beams to the plasma. An amount of neutral beam power is determined by the particles of a certain energy at steady state as following;

$$F_{pla_nb_d}(t) = \frac{3.6 \cdot 10^3}{1.6} \cdot 10^9 \cdot \frac{1}{N_o} \cdot \frac{f_{power} \cdot W_{_nb}}{E_{_nb}} \tag{2.65}$$

were,

$F_{pla_nb_d}(t)$: flow of neutral beam injection (mole/h)

N_o : Avogadro's Number

f_{power} : normalization factor of simulated pulse operation (defined by Fig. 2.3)

$W_{_nb}$: neutral beam power (75MW)

$E_{_nb}$: neutral beam particle energy (1.3MeV).

Fig. 2.56 shows the flow diagram of the neutral beam injection subsystem.

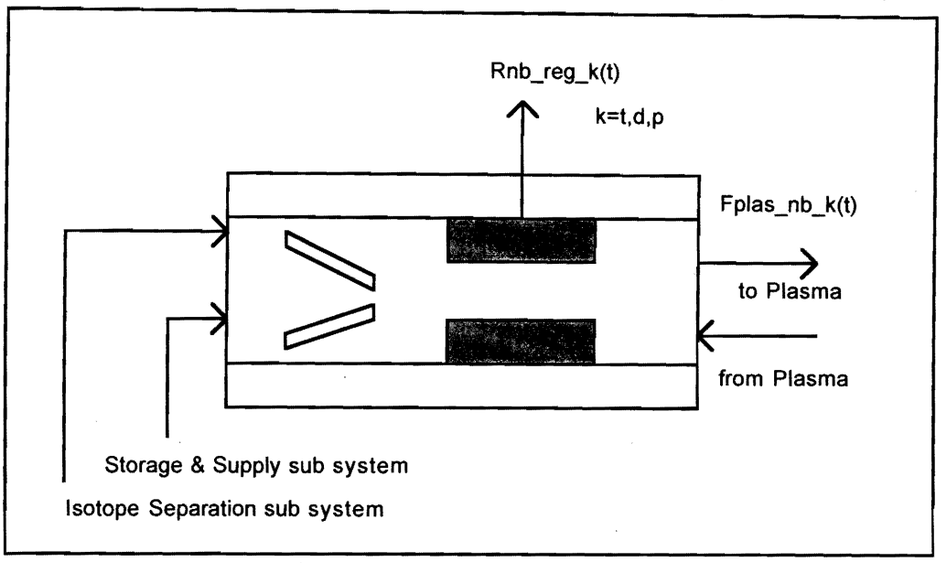


Fig. 2.56 Flow diagram of the neutral beam injection subsystem

(1) The control equation

The neutral beam is supplemented by a delivery from the deuterium rich stream of the isotope separation subsystem and the deuterium rich stream of the storage and supply subsystem. These flows provide the beam flow requirement. The stream of the storage and supply subsystem supplements to meet the required flow on the case when the stream from the isotope separation subsystem be in insufficient. The flow of the neutral beam injection to the plasma chamber could be expressed by the following expressions;

$$\frac{F_{plas_nb_d}(t)}{\varepsilon_{nb_tot}} = F_{in_st_d}(t) \cdot \chi_{sto_deu_d}(t) + k_{iss_mid} \cdot F_{iss_mid}(t) \cdot \chi_{iss_mid_d}(t) \tag{2.66}$$

where,

$F_{in_st_d}(t)$: flow of deuterium from the storage and supply subsystem (mole/h)

$F_{iss_mid}(t)$: flow of deuterium from middle of the isotope separation subsystem (mole/h)

$\chi_{sto_deu_d}(t)$: mole fraction of deuterium in the deuterium rich stream of the storage and supply subsystem

$\chi_{iss_mid_d}(t)$: mole fraction of deuterium in the deuterium rich stream of the isotope separation subsystem

k_{iss_mid} : fraction of flow from middle of the isotope separation subsystem

ε_{nb_tot} : efficiency of the neutral beam injection (0.3).

Maintaining a high vacuum ambiance is needed for the beam injection in the neutral beam ports. The variances in the time of the space vector components in the process of continuous regeneration of the cryopumps to maintain high vacuum ambiance could be expressed by the following expressions;

$$\frac{d}{dt}\begin{bmatrix} N_{cyo_}(t) \\ T_{nb_cryo}(t) \\ D_{nb_cryo}(t) \end{bmatrix} = f_{cryo_}(t)\cdot k_{iss_mid}\cdot F_{iss_mid}(t)\cdot \begin{bmatrix} 1 \\ \chi_{iss_mid_t} \\ \chi_{iss_mid_d} \end{bmatrix} + f_{cyo_}(t)\cdot F_{in_sto_d}\cdot \begin{bmatrix} 1 \\ \chi_{sto_deu_t} \\ \chi_{sto_deu_d} \end{bmatrix} - \frac{\varepsilon_{scrap}\cdot r_{scrap}}{a_{cryo}} \begin{bmatrix} N_{nb_}(t) \\ T_{nb_cryo}(t) \\ D_{nb_cryo}(t) \end{bmatrix} \tag{2.67}$$

$$P_{nb_cryo}(t) = \chi_{nb_cryo_p}(t)\cdot N_{nb_cryo}(t) \tag{2.68}$$

$$\chi_{nb_cryo_t}(t) = \frac{T_{nb_cryo}(t)}{N_{cryo_}(t)} \tag{2.69}$$

$$\chi_{nb_cryo_d}(t) = \frac{D_{nb_cryo}(t)}{N_{cryo_}(t)} \tag{2.70}$$

$$\chi_{nb_cryo_p}(t) = 1 - \chi_{nb_cryo_t}(t) - \chi_{nb_cryo_d}(t) \tag{2.71}$$

where,

$N_{cyo_}(t)$: total amount of particles on cryopump (moles)

$T_{nb_cryo}(t)$: tritium on cryopump (moles)

$D_{nb_cryo}(t)$: deuterium on cryopump (moles)

$P_{nb_cryo}(t)$: protium on cryopump (moles)

$F_{iss_mid}(t)$: flow from middle of the isotope separation subsystem (mole/h)

$F_{in_sto_d}$: flow of the deuterium rich stream from the storage and supply subsystem (mole/h)

$f_{cryo_}(t)$: fraction of feed to neutral beam that flows to cryopump

(0.37)

k_{iss_mid} : fraction of flow from middle of the isotope separation subsystem directed to the neutral beam injection (0.3)

ε_{scrap} : removal efficiency of cryodeposit (0.2)

r_{scrap} : scraping rate on cryosurface (1.92m2/h)

a_{cryo} : cryosurface per pump (0.032m2)

$\chi_{iss_mid_t}$: mole fraction of tritium from middle of the isotope separation subsystem

$\chi_{iss_mid_d}$: mole fraction of deuterium from middle of the isotope separation subsystem

$\chi_{sto_deu_t}$: mole fraction of tritium in the deuterium rich stream from the storage and supply subsystem

$\chi_{sto_deu_d}$: mole fraction of deuterium in the deuterium rich stream from the storage and supply subsystem

$\chi_{nb_cryo_p}(t)$: mole fraction of protium on cryopump.

(2) The observation equation

On the continuous regeneration process the observation equation could be established by deriving the corresponding term defined by equation (2.67);

$$\mathbf{y} = \frac{\varepsilon_{scrap} \cdot r_{scrap}}{a_{cryo}} \cdot \begin{bmatrix} T_{nb_cryo}(t) \\ D_{nb_cryo}(t) \\ P_{nb_cryo}(t) \end{bmatrix} = \begin{bmatrix} R_{nb_reg_t}(t) \\ R_{nb_reg_d}(t) \\ R_{nb_reg_p}(t) \end{bmatrix} \quad (2.72).$$

The observation vector components, $R_{nb_reg_t}(t)$, $R_{nb_reg_d}(t)$, $R_{nb_reg_p}(t)$, defined by equation (2.72) are delivered to the isotope separation subsystem. The required flow rate of the neutral beam injection into the plasma chamber was defined by the equation (2.66) using the efficiency of neutral beam injection. Using the mole fraction coefficients in the stream, each particle flow of the neutral beam injection into the plasma chamber could be expressed by the following expressions;

$$\begin{bmatrix} F_{plas_nb_}(t) \\ F_{plas_nb_t}(t) \\ F_{plas_nb_d}(t) \end{bmatrix} = k_{iss_mid} \cdot F_{iss_mid}(t) \cdot \begin{bmatrix} 1 \\ \chi_{iss_mid_t} \\ \chi_{iss_mid_d} \end{bmatrix} \cdot \varepsilon_{nb_tot} + F_{in_st_d} \cdot \begin{bmatrix} 1 \\ \chi_{sto_deu_t} \\ \chi_{sto_deu_d} \end{bmatrix} \cdot \varepsilon_{nb_tot} \quad (2.73)$$

$$F_{plas_nb_p}(t) = F_{plas_nb_}(t) - F_{plas_nb_t}(t) - F_{plas_nb_d}(t) \quad (2.74).$$

The simulation results of the observation vector components, $R_{nb_reg_t}(t)$, $R_{nb_reg_d}(t)$, $R_{nb_reg_p}(t)$, defined by equation (2.72) are described in the Fig.2.57, Fig. 2.58, Fig. 2.59, respectively.

The simulation results of the observation vector components, $F_{plas_nb_t}(t)$, $F_{plas_nb_d}(t)$ and $F_{plas_nb_p}(t)$, defined by equation (2.73) and (2.74) are described in the Fig. 2.60, Fig. 2.61, Fig. 2.62, respectively.

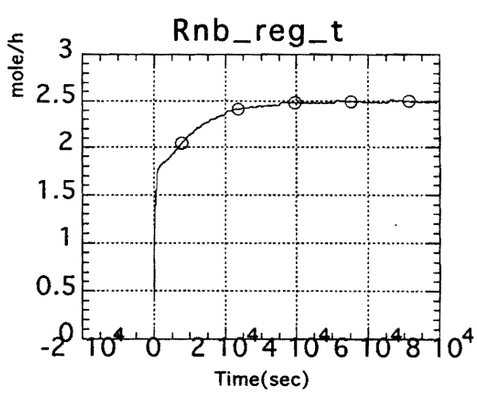


Fig. 2.57 Regeneration rate of tritium from the neutral beam injection cryopump (mole/h) : Rnb_reg_t(t)

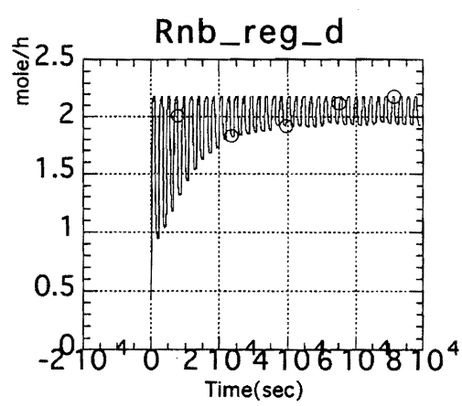


Fig. 5.58 Regeneration rate of deuterium from the neutral beam injection cryopump (mole/h) : Rnb_reg_d(t)

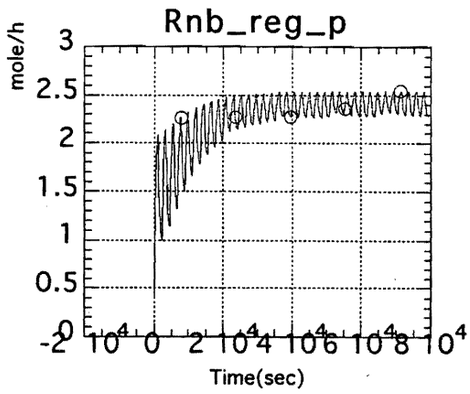


Fig. 2.59 Regeneration rate of protium from the neutral beam injection cryopump (mole/h) : Rnb_reg_p(t)

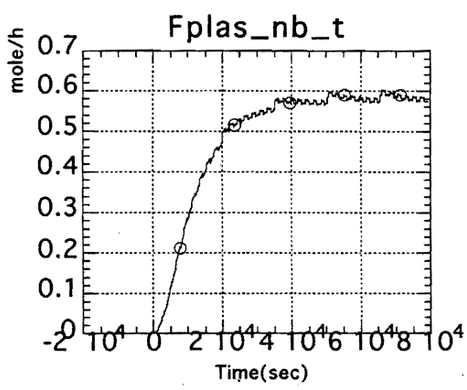


Fig. 2.60 Flow of tritium to the plasma chamber through the neutral beam injection (mole/h) : Fplas_nb_t(t)

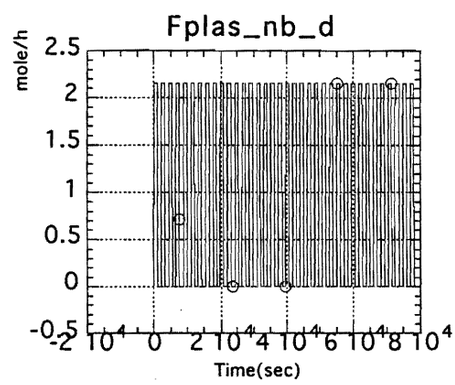


Fig. 2.61 Flow of deuterium to the plasma chamber through the neutral beam injection (mole/h) : Fplas_nb_d(t)

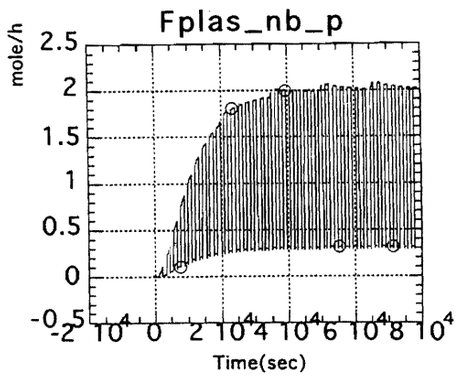


Fig. 2.62 Flow of protium to the plasma chamber through the neutral beam injection (mole/h) : Fplas_nb_p(t)

3.1.6 Pellet Injector Subsystem

Pellet injectors provide core fueling to achieve and maintain ignited plasmas. Three pellet injectors are designed for operation phases with a variable ratio of tritium and deuterium feed gases [1]. On this simulation model a single stage pellet injector model is adopted for continuous fueling well beyond the edge region during the extended operation phases. Fig. 2.63 shows the single stage pellet injector simulation model. Feed to the extruders is supplied from the various sources including the isotope separation, the fuel clean-up and the storage and supply subsystem. The propellant source is protium which is provided by the feeds from the isotope separation and the storage and supply subsystem.

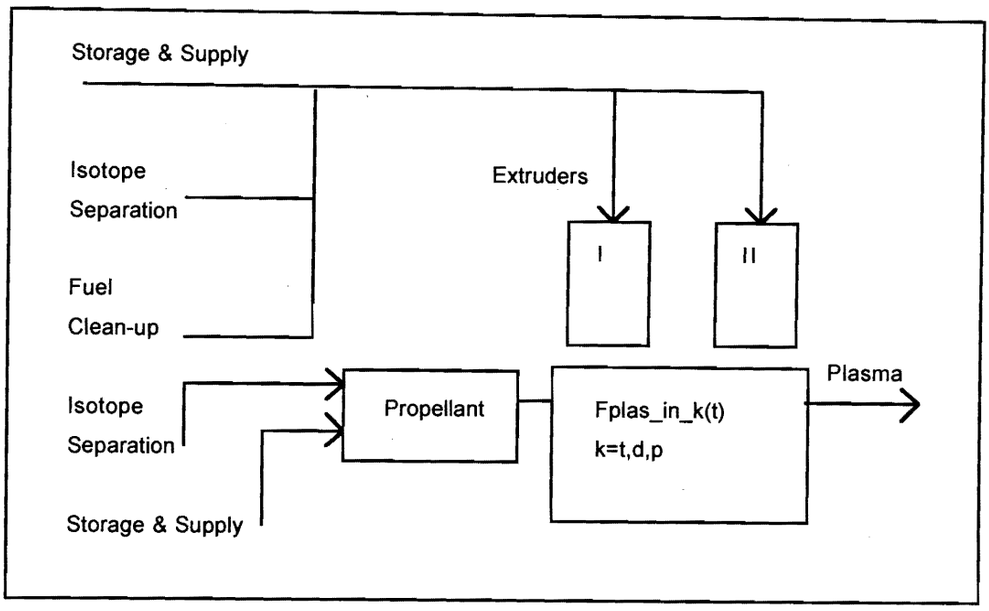


Fig. 2.63 Flow diagram of pellet injector subsystem

(1) The observation equation

The pellet injector works on the projectile using frozen hydrogen pellets encased in sabots. For continuous fueling, two extruders are shared so that one be supplying pellets, other be forming ices. Feed to the extruders is supplied from the sources including the isotope separation, the fuel clean-up and the storage and supply subsystem. Initially, extruder I would be with full inventory, while extruder II would be empty. On this initial conditions, extruder I would become in supplying pellets, and extruder II would be getting in supply from the sources including the isotope separation, the fuel clean-up and the storage and supply subsystem. Then the observation equation could be expressed by the following expressions;

$$\frac{1}{(1-\varepsilon_{inj_err})}\begin{bmatrix} F_{plas_in_t}(t) \\ F_{plas_in_d}(t) \end{bmatrix} = F_{sto_out_t}(t)\cdot\begin{bmatrix} \chi_{sto_tri_t}(t) \\ \chi_{sto_tri_d}(t) \end{bmatrix}$$
$$+F_{sto_out_d}(t)\cdot\begin{bmatrix} \chi_{sto_deu_t}(t) \\ \chi_{sto_deu_d}(t) \end{bmatrix}$$
$$+h_3\cdot F_{puri_out}(t)\cdot\begin{bmatrix} \chi_{iss_in_t}(t) \\ \chi_{iss_in_d}(t) \end{bmatrix} \tag{2.75}$$
$$+l_{inj_mid_iss}\cdot F_{iss_mid}(t)\cdot\begin{bmatrix} \chi_{iss_mid_t}(t) \\ \chi_{iss_mid_d}(t) \end{bmatrix}$$
$$+l_{inj_bot_iss}\cdot F_{iss_bot}(t)\cdot\begin{bmatrix} \chi_{iss_bot_t}(t) \\ \chi_{iss_bot_d}(t) \end{bmatrix}$$

where,

$F_{plas_in_t}(t)$: flow of tritium to the plasma through the pellet injector subsystem(mole/h)

$F_{plas_in_d}(t)$: flow of deuterium to the plasma through the pellet injector (mole/h)

$F_{sto_out_t}(t)$: flow of the tritium rich stream from the storage and supply subsystem (mole/h)

$F_{sto_out_d}(t)$: flow of the deuterium rich stream from the storage and supply subsystem (mole/h)

$F_{puri_out}(t)$: total outlet flow from the fuel clean-up subsystem (mole/h)

$F_{iss_mid}(t)$: flow from middle of the isotope separation subsystem (mole/h)

$F_{iss_bot}(t)$: flow from bottom of the isotope separation subsystem (mole/h)

ε_{inj_err} : erosion fraction of deuterium and tritium for the pellet injector (0.3)

h_3 : fraction of stream not processed and directed to the pellet injector subsystem (0.3)

$l_{inj_mid_iss}$: fraction of flow from middle of the isotope separation subsystem directed to the pellet injector subsystem (0.3)

$l_{inj_bot_iss}$: fraction of flow from bottom of the isotope separation subsystem directed to the pellet injector subsystem (0.3)

$\chi_{sto_tri_t}(t)$: mole fraction of tritium in the tritium rich stream from the storage and supply subsystem

$\chi_{sto_tri_d}(t)$: mole fraction of deuterium in the tritium rich stream from the storage and supply subsystem

$\chi_{sto_deu_t}(t)$: mole fraction of tritium in the deuterium rich stream from the storage and supply subsystem

$\chi_{sto_deu_d}(t)$: mole fraction of deuterium in the deuterium rich stream from the storage and supply subsystem

$\chi_{iss_in_t}(t)$: mole fraction of tritium from the fuel clean-up subsystem

$\chi_{iss_in_d}(t)$: mole fraction of deuterium from the fuel clean-up subsystem

$\chi_{iss_mid_t}(t)$: mole fraction of tritium from middle of the isotope separation subsystem

$\chi_{iss_mid_d}(t)$: mole fraction of deuterium from middle of the isotope separation subsystem

$\chi_{iss_bot_t}(t)$: mole fraction of tritium from bottom of the isotope separation subsystem

$\chi_{iss_bot_d}(t)$: mole fraction of deuterium from middle of the isotope separation subsystem.

The simulation results of the observation vector components, $F_{plas_in_t}(t)$, $F_{plas_in_d}(t)$, defined by the equation (2.75) are described in the Fig. 2.64, Fig. 2.65, respectively.

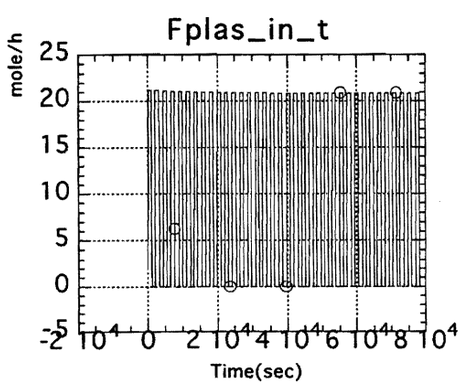


Fig. 2.64 Flow of tritium through the pellet injector (mole/h) : Fplas_in_t(t)

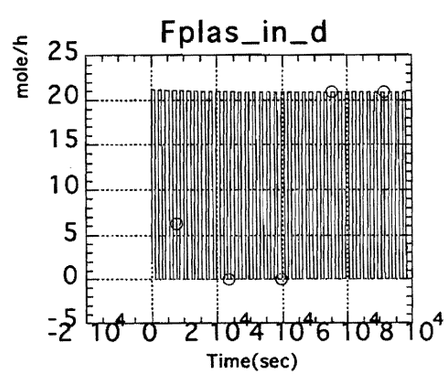


Fig. 2.65 Flow of deuterium through the pellet injector (mole/h) : Fplas_in_d(t)

The feed rate to the extruder to be filled must be met to the required flow rate to the plasma. Assuming that mole requirement per propellant is

constant, repetition rate of injector could be introduced by the following relations;.

$$R_{inj_}(t) = \frac{F_{plas_in_t}(t)}{n_{pellet_t}(1-\varepsilon_{inj_err})} \tag{2.76}$$

where,

$R_{inj_}(t)$: pellet repetition rate for pellet injector (Hz)

n_{pellet_t} : number of moles in pellet injector pellet.

The propellant requirement is supposed to have a fixed number of moles to be used to propel the projectile. The propellant requirement could be expressed by the following expressions;

$$F_{inj_prop}(t) = n_{prop} \cdot R_{inj_}(t) \tag{2.77}$$

$$F_{inj_prop}(t) = F_{sto_out_p}(t) + l_{iss_p} \cdot F_{iss_top_}(t) \tag{2.78}$$

$$F_{sto_out_p}(t) = F_{inj_prop}(t) - l_{iss_p} \cdot F_{iss_top_}(t) \tag{2.79}$$

where,

$F_{inj_prop}(t)$: flow of propellant of pellet injector (mole/h)

$F_{sto_out_p}(t)$: flow of protium rich stream from the storage and supply subsystem (mole/h)

n_{prop} : amount of propellant gas required per pellet for pellet injector (0.4 moles)

l_{iss_p} : fraction of flow from top of the isotope separation system directed to the pellet injector subsystem (0.03).

The simulation results of the repetition rate of injector, $R_{inj_}(t)$, and propellant flow, $F_{inj_prop}(t)$, are described in the Fig. 2. 66 and Fig. 2.67, respectively.

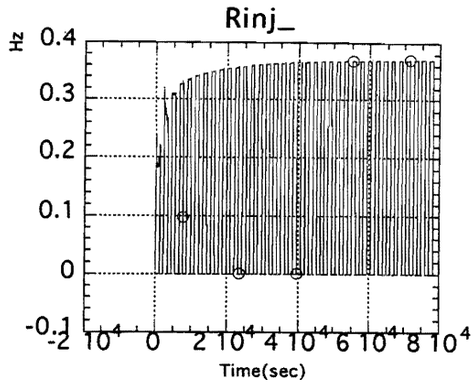


Fig. 2.66 Pellet repetition rate for pellet injector (Hz) : Rinj_(t)

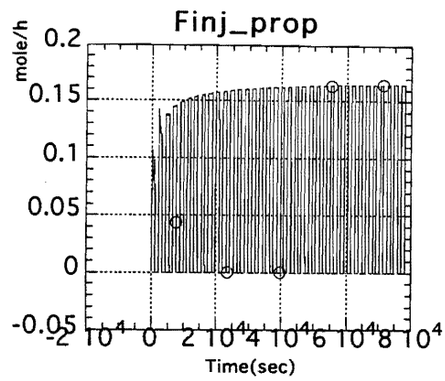


Fig. 2.67 Flow of propellant of pellet injector (mole/h) : Finj_prop(t)

3.1.7 Gas Puffer Subsystem

The gas puffer subsystem has been implemented for fueling the plasma edge, not the plasma core. Feed to the gas puffer subsystem is supplemented from the isotope separation , the fuel clean-up, and the storage and supply subsystem. The gas puffer subsystem consists of reservoir tank. Fig. 2.68 shows the flow diagram of the gas puffer subsystem.

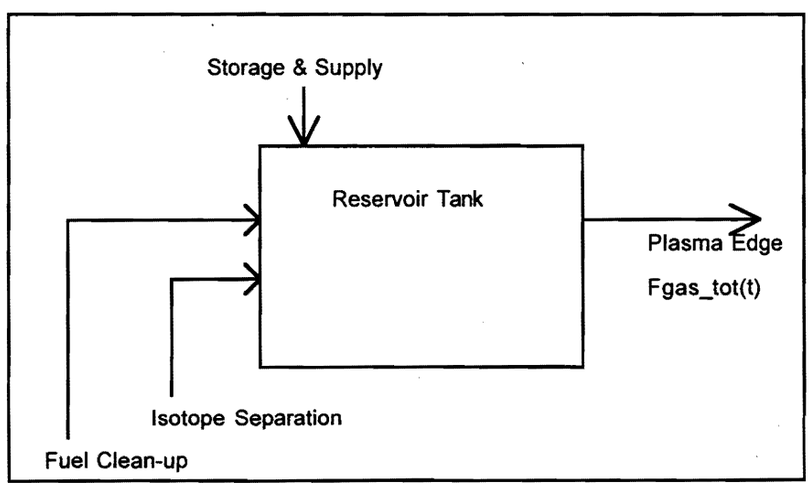


Fig. 2.68 Flow diagram of the gas puffer subsystem

(1) The observation equation

The specified flow to the plasma edge is provided by the feed from the isotope separation , the fuel clean-up, and the storage and supply subsystem. The observation equation could be expressed by the following expressions;

$$\begin{bmatrix} F_{gas_t}(t) \\ F_{gas_d}(t) \end{bmatrix} = F_{gas_in_tri}(t) \cdot \begin{bmatrix} \chi_{sto_tri_t}(t) \\ \chi_{sto_tri_d}(t) \end{bmatrix} + F_{gas_in_deu}(t) \cdot \begin{bmatrix} \chi_{sto_deu_t}(t) \\ \chi_{sto_deu_d}(t) \end{bmatrix}$$
$$+ h_1 \cdot F_{puri_out}(t) \cdot \begin{bmatrix} \chi_{iss_in_t}(t) \\ \chi_{iss_in_d}(t) \end{bmatrix}$$
$$+ m_{gas_mid_iss} \cdot F_{iss_mid}(t) \cdot \begin{bmatrix} \chi_{iss_mid_t}(t) \\ \chi_{iss_mid_d}(t) \end{bmatrix} \tag{2.80}$$
$$+ m_{gas_bot_iss} \cdot F_{iss_bot}(t) \cdot \begin{bmatrix} \chi_{iss_bot_t}(t) \\ \chi_{iss_bot_d}(t) \end{bmatrix}$$

where,

$F_{gas_t}(t)$: tritium flow to the plasma edge from the gas puffer subsystem (mole/h)

$F_{gas_d}(t)$: deuterium flow to the plasma edge from the gas puffer subsystem (mole/h)

$F_{gas_in_tri}(t)$: flow from the tritium rich stream from the storage and supply subsystem (mole/h)

$F_{gas_in_deu}(t)$: flow from the deuterium rich stream from the storage and supply subsystem (mole/h)

$F_{puri_out}(t)$: total outlet flow from the fuel clean-up subsystem (mole/h)

$F_{iss_mid}(t)$: flow from middle of the isotope separation subsystem (mole/h)

$F_{iss_bot}(t)$: flow from bottom of the isotope separation subsystem (mole/h)

h_1 : fraction of stream not processed and directed to the gas puffer subsystem (0.3)

$m_{gas_mid_iss}$: fraction of flow from middle of the isotope separation subsystem directed to the gas puffer subsystem (0.3)

$m_{gas_bot_iss}$: fraction of flow from bottom of the isotope separation subsystem directed to the gas puffer subsystem (0.3)

$\chi_{sto_tri_t}(t)$: mole fraction of tritium in the tritium rich stream from the storage and supply subsystem (mole/h)

$\chi_{sto_tri_d}(t)$: mole fraction of deuterium in the tritium rich stream from

the storage and supply subsystem (mole/h)

$\chi_{sto_deu_t}(t)$: mole fraction of deuterium in the deuterium rich stream from the storage and supply subsystem (mole/h)

$\chi_{sto_deu_d}(t)$: mole fraction of deuterium in the deuterium rich stream from the storage and supply subsystem (mole/h)

$\chi_{iss_in_t}(t)$: mole fraction of tritium in the stream from the fuel clean-up subsystem

$\chi_{iss_in_d}(t)$: mole fraction of deuterium in the stream from the fuel clean-up subsystem

$\chi_{iss_mid_t}(t)$: mole fraction of tritium in the stream from middle of the isotope separation subsystem

$\chi_{iss_mid_d}(t)$: mole fraction of deuterium in the stream from middle of the isotope separation subsystem

$\chi_{iss_bot_t}(t)$: mole fraction of tritium in the stream from bottom of the isotope separation subsystem

$\chi_{iss_bot_d}(t)$: mole fraction of deuterium in the stream from bottom of the isotope separation subsystem.

The simulation results of the observation vector components, $F_{gas_t}(t)$, $F_{gas_d}(t)$, defined by the equations (2.80) are described in the Fig. 2.69, Fig. 2.70, respectively.

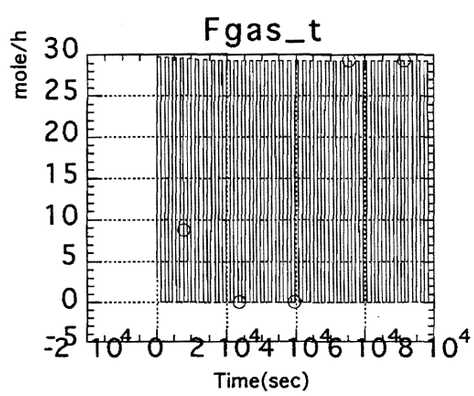


Fig. 2.69 Flow of tritium through the gas puffer (mole/h): Fgas_t(t)

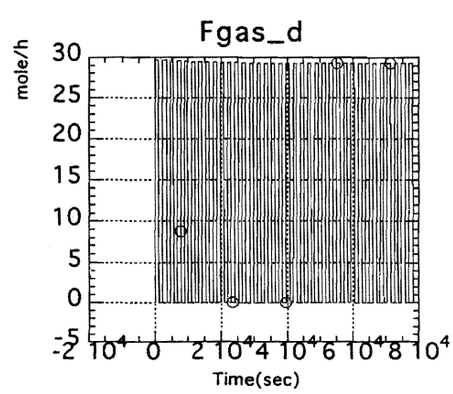


Fig. 2.70 Flow of deuterium through the gas puffer (mole/h) : Fgas_d(t)

3.1.8 Storage and Supply Subsystem

All initial inventory must be previously supplemented on the storage and supply subsystem before operation of fuel cycle is launched. During the operation of fuel cycle, the storage and supply subsystem will provide the amount of supplemental tritium, deuterium, and protium to meet the required flows which are specified according to the progress of time over the fuel processing. The storage and supply subsystem consists of three kinds of storage systems, tritium rich, deuterium rich, and protium rich ones. Fig. 2.71 shows the flow diagram of the storage and supply subsystem.

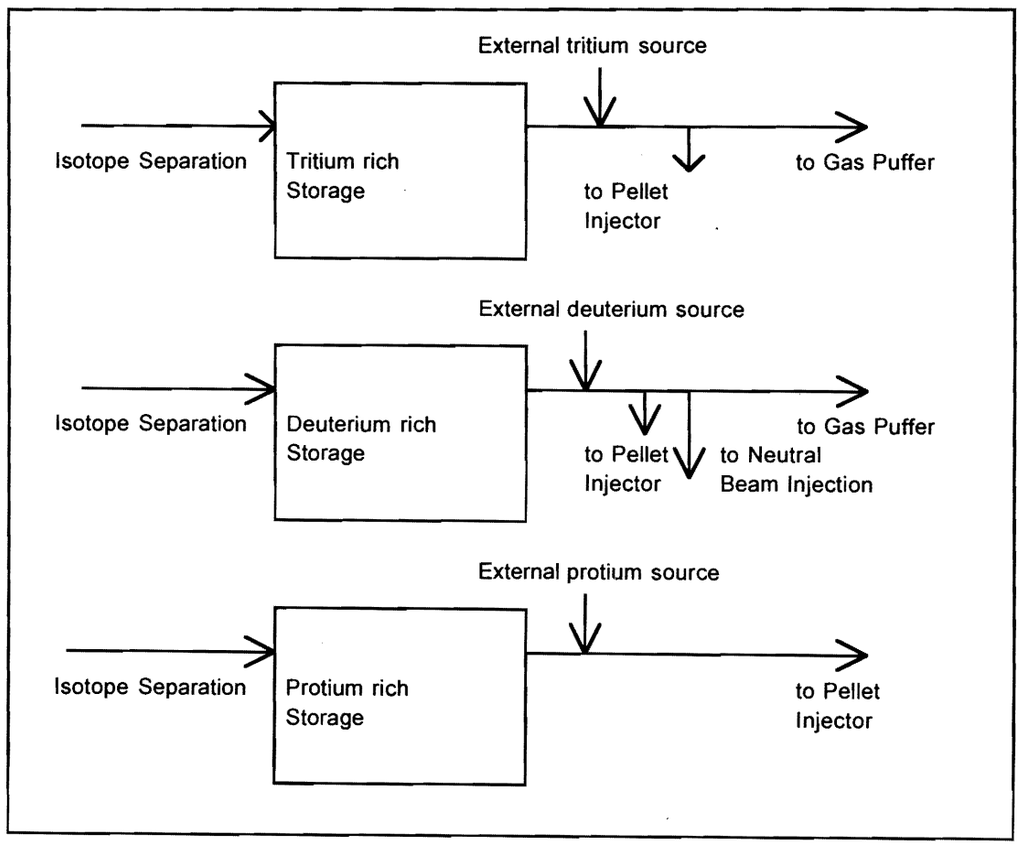


Fig. 2.71 Flow diagram of the storage and supply subsystem

(1) The control equation

An account of supplemental tritium, deuterium, and protium in the three kinds of storage systems could be formulated by the following relations;

$$\frac{d}{dt}\begin{bmatrix} N_{sto_tri_}(t) \\ N_{sto_deu_}(t) \\ N_{sto_pro_}(t) \end{bmatrix} = \begin{bmatrix} F_{sto_tri_in}(t) \\ F_{sto_deu_in}(t) \\ F_{sto_pro_in}(t) \end{bmatrix} - \begin{bmatrix} F_{sto_tri_out}(t) \\ F_{sto_deu_out}(t) \\ F_{sto_pro_out}(t) \end{bmatrix} \tag{2.81}$$

where,

$N_{sto_tri_}(t)$: total amount in the tritium rich storage (moles)

$N_{sto_deu_}(t)$: total amount in the deuterium rich storage (moles)

$N_{sto_pro_}(t)$: total amount in the protium rich storage (moles)

$F_{sto_tri_in}(t)$: flow into the tritium rich storage (mole/h)

$F_{sto_deu_in}(t)$: flow into the deuterium rich storage (mole/h)

$F_{sto_pro_in}(t)$: flow into the protium rich storage (mole/h0

$F_{sto_tri_out}(t)$: flow out of the tritium rich storage (mole/h)

$F_{sto_deu_out}(t)$: flow out of the deuterium rich storage (mole/h)

$F_{sto_pro_out}(t)$: flow out of the protium rich storage (mole/h).

The variance in the time of the total amount of tritium in the tritium rich storage could be expressed by the following expressions;

$$\frac{dT_{sto_tri_}(t)}{dt} = F_{sto_tri_in}(t)\cdot\chi_{iss_bot_t}(t) - F_{sto_tri_out}(t)\cdot\frac{T_{sto_tri_}(t)}{N_{sto_tri_}(t)} \tag{2.82}$$

where,

$T_{sto_tri_}(t)$: total amount of tritium in tritium rich storage (moles)

$\chi_{iss_bot_t}(t)$: mole fraction of tritium in the stream from bottom of the isotope separation subsystem.

The variance in the time of the tritium inventory in the tritium rich storage defined by the following expressions;

$$K_{sto_tri_}(t) = \sum_{i=1}^{t}\left\{T_{sto_tri_}(i-1) - T_{sto_tri_}(i)\right\} \tag{2.83}$$

where,

$K_{sto_tri_}(t)$: varience in the time of tritium inventory in the tritium rich storage (moles).

The simulation results of the state vector components defined by the equation (2.82) and (2.83) described in the Fig. 2. 72 and Fig. 2.73.

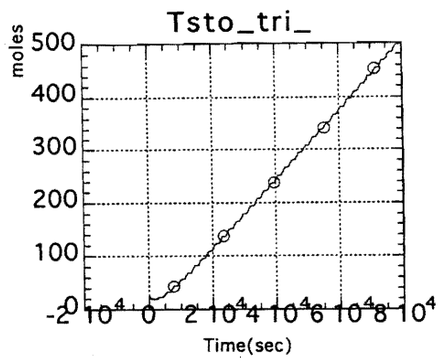


Fig. 2.72 Total amount of tritium in the tritium rich storage (moles) :Tsto_tri_(t)

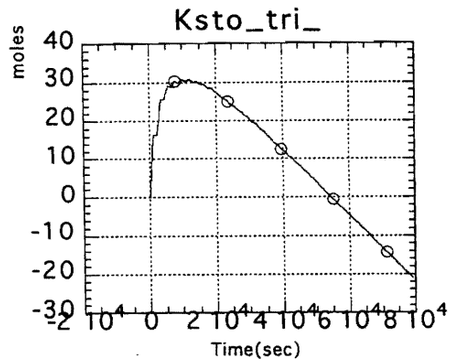
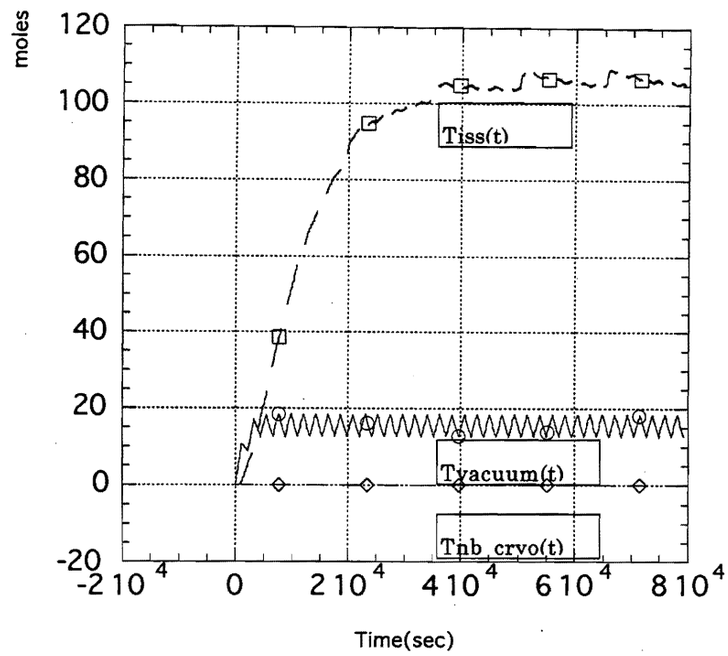


Fig. 2.73 Varience in the time of the inventory of tritium in the tritium rich storage (moles) : Ksto_tri_(t)

3.2 Discussions

(1) Tritium inventories

The results show that the tritium inventories in the fuel processing will vary depending upon plasma operation scenario and individual subsystem operation designs. Individual subsystem is basically unit operation working with their processing parameters. Consequently, the observation vector in the equation (2.2) whose components denote the variance in the time of inventories vary depending on their operation cycles. The tritium inventories in the subsystems of the fuel processing system forced by the plasma chamber exhaust flow are traced for a 2 day simulation period. Fig. 3.1 shows localized tritium inventories in the major subsystems for 36 pulse operation cycles. The most part of tritium inventories can be seen to be located in 2 subsystems; the vacuum pump subsystem and the isotope separation subsystem. The remaining inventory is localized in the neutral beam injection subsystem.



Tiss(t) : tritium in isotope separation subsystem (moles)
Tvacuum(t) : tritium in vacuum pump subsystem (moles)
Tnb_cryo(t) : tritium in neutral beam injection (moles)

Fig. 3.1 Tritium inventories

a. The vacuum pump subsystem

About 17% of tritium inventory is localized in the vacuum pump subsystem. The tritium inventory in the vacuum pump subsystem can be seen to vary depending upon the cryopumps arrangement and pump cycle description. The cryopumps arrangement and pump cycle description during partial regeneration is described in the Fig. 2.16. The variance in the time of tritium inventory in the period of 3 pulses operation is described in Fig. 3.2. This figure shows the fact that the inventory fluctuation in the vacuum pump subsystem has been arisen by 2000 sec periodical interval. The gradients of the rising and the falling in the inventory are corresponding to the average time which is determined from the 24 cryopumps arrangement and pump cycle description described in the Fig. 2.16.

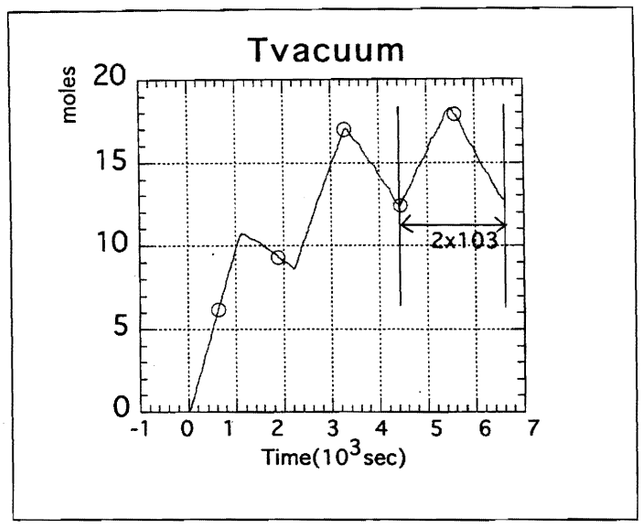


Fig.3.2 The variance in the time of inventory of the vacuum pump subsystem in the period of 3 pulses operation

b. The isotope separation subsystem

About 83% of tritium inventory is localized in the isotope separation subsystem. A large amounts of accumulative tritium inventory in the isotope separation subsystem has been caused by a taking of its overall residence time, 2.22h, larger than any subsystem else. The variance in the time of inventory of the isotope separation subsystem can be seen relatively to be moderated in spite of getting a frequently fluctuating inlet flow. There are the discontinuities which arise on the variances in the time of inventory with the periodical uprise having a regular interval (150000sec). The interval corresponds to the time when the pulsing in the flow out of upstream of the isotope separation subsystem is appeared. The pulsing arises in the stream of the fuel clean-up subsystem when the standby bed is changed from the preloading to the purification according to the progress of time in fuel clean-up processing (Fig. 2.43).

c. The neutral beam injection subsystem

The tritium in the process of continuos regeneration of the cryopumps is expressed in the equation (2.67). The variance in the time of tritium inventory on the cryopumps in the neutral beam injection subsystem is described in Fig. 3.3.

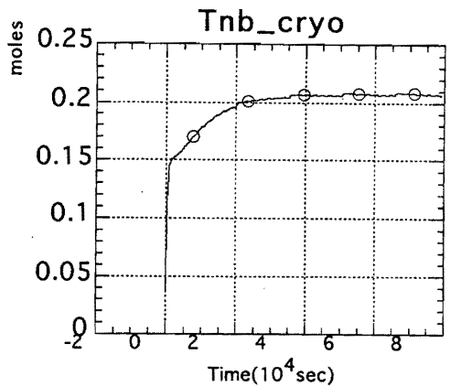


Fig. 3.3 Time variance of tritium inventory in the neutral beam injection (moles) : Tnb_cryo(t)

d. The storage and supply subsystem

The storage and supply subsystem provides an amount of supplemental tritium to meet the required flow corresponding to the fuel processing. The storage and supply subsystem provides the only sources of tritium because the product stream out of the isotope separation subsystem are not being discharged yet at beginning of pulse operation. An imbalance caused on the gaps between upstream processing flow and required flow is supplemented by the feed from external sources. Fig. 2.72 shows that the amount of tritium in tritium rich storage, $T_{sto_tri_}(t)$, is a monotone increasing function except a period at beginning of pulse operation. This fact yields denominator of the right hand second term in the equation (2.82), $N_{sto_tri_}(t)$, is holding a negative sign. A negative sign of $N_{sto_tri_}(t)$ denotes an imbalance resulting from the decreasing of total amount in the tritium rich storage while upstream processing not able to balance it with hold up of incoming flow. Fig. 2.72 shows the fact that the total amount of tritium in the tritium rich storage reaches to 500 moles at the end of the 36 pulse operation cycles. The total amount of tritium in the tritium rich storage reaches to 1100 moles at end of the 79 pulse operation cycles (2 days). Fig. 3.4 shows the total amount of tritium in the tritium rich storage over the 79 pulse operation cycles.

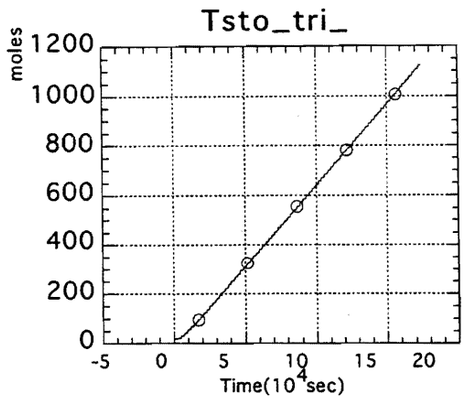


Fig. 3.4 The total amount of tritium in the tritium rich storage (moles) Tsto_tri_(t)

Fig. 2.73 shows the fact that the tritium inventory in the tritium rich storage reaches to 20 moles at the end of 36 pulse operation cycles. The tritium inventory in the tritium rich storage reaches to 100 moles at the end of 79 pulse operation cycles (2 days). Fig. 3.5 shows the variance in the time of tritium inventory in the tritium rich storage over the 79 pulse operation.

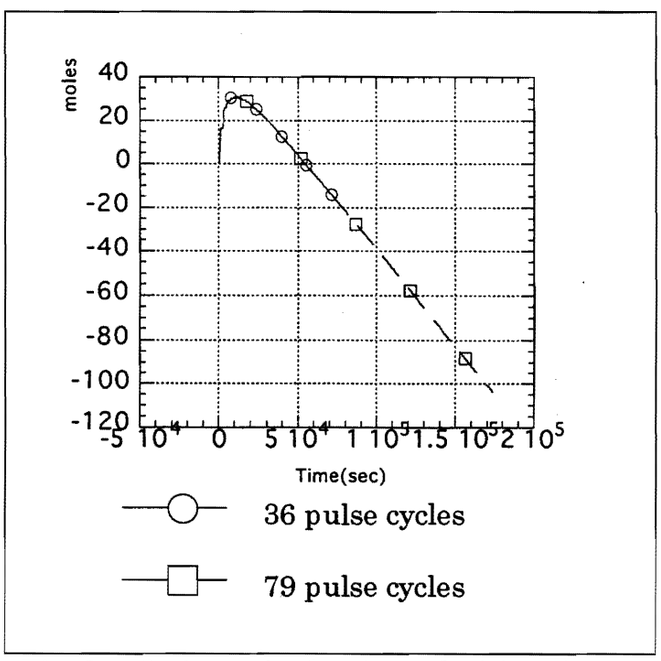


Fig. 3.5 Variance in the time of tritium inventory in the tritium rich storage

4. Summary

To fully realize the state-space description of the fuel processing cycles forced by the plasma chamber exhaust flows, the linear ordinary differential equations for subsystems have been formulated. The state-space descriptions formulated in the linear ordinary differential equation were simulated over the pulse operation cycles lasting 2 days. This simulation code has been shown to be useful for determining the dynamic behavior of subsystems forced by the plasma chamber exhaust flows. Results have been obtained which elucidate the dynamic behavior of subsystems. The requirement of better understanding of safe and reliable operation of the fuel processing is the controllability in the time-variance governing the processing with regard to tritium inventory. The variance in the time of tritium inventories in the fuel processing has been shown.

Acknowledgment

This work has been performed as one of the activities of the Fusion Reactor System Laboratory. The authors gratefully acknowledge S. Ueda, R. Kurihara, S. Nishio, T. Ajima of JAERI for their contributions to perform this work.

References

[1] ITER Fuel Cycle, ITER Documentation Series, No. 31, IAEA, Vienna, 1991.

[2] W. Kuan, M.A. Abdou, and R.S. Willms, Dynamic Simulation of a Proposed ITER Tritium Processing System, Fusion Technology, Vol. 28, 1995.

[3] W. Kuan, M.A. Abdou, and R.S. Willms, Time-Dependent Tritium Inventories and Flow Rates in Fuel Cycle Components of a Tokamak Fusion Reactor, Fusion Engineeringand Design, Vol. 28, 1995.

[4] C. K. Chui and G. Chen, Linear Systems and Optimal Control, Springer-Verlag, 1988.

[5] R. T. McGrath et al., Design Consideration for ITER Plasma Facing Components, SAND 89-0901, July 1989.

[6] B. Friedland, Linear Systems, McGraw-Hill Book Company, 1965.

Appendix Source List
page 1

● _main.f

```
 1:c
 2:c ***************************************************************
 3:c
 4:c     Fusion reactor tritium inventory evaluation system
 5:c
 6:c     By the Japan Atomic Energy Research Institute, 1997
 7:c
 8:c ***************************************************************
 9:c
10:c-----------------------------------------------------------------
11:c $Id: _main.f,v 1.8 1998/03/26 05:22:22 sasaki Exp $
12:c-----------------------------------------------------------------
13:c
14:c    Main routine
15:c
16:c-----------------------------------------------------------------
17:c
18:     program main
19:c
20:     implicit real*8 ( a-h, o-z )
21:c
22:     include '_sizes.h'
23:     include '_commons1.h'
24:c
25:     include '_storage.h'
26:c
27:     include 'varconst.h'
28:c
29:     integer errsw
30:     character*80 line
31:c
32:     nvar = 291
33:     nconst = 99
34:
35:c-----------------------------------------------------------------
36:c  System dependent initialization
37:c   this may includes ;
38:c     * Options (from command line, menus, or files)
39:c     * File name & I/O unit allocation. (if necessary)
40:c     * Menu control (if possible)
41:c     * Setting floating point hardware, exception handling etc.
42:c       (if necessary).
43:c
44:c     etc.
45:c
46:c    Returns system name such as 'Macintosh'
47:c-----------------------------------------------------------------
48:c
49:     call INITPROG(sysname)
50:c
51:c-----------------------------------------------------------------
52:c  Preprocess input and put lines on unit 50
53:c-----------------------------------------------------------------
54:c
55:     open(unit=50,status='SCRATCH',form='FORMATTED')
56:  10 line = ' '
57:     read(5,FMT='(a)',end=20) line
58:c
59:     if( line(1:1).ne.'#' .and. line.ne.' ' ) then
60:        write(50,'(a)') line
61:     end if
62:     goto 10
63:c
64:  20 rewind(50)
65:c
66:c-----------------------------------------------------------------
67:c  Input file name specification and options etc.:
68:c-----------------------------------------------------------------
69:c
70:     call FILENAMEINPUT
71:c
72:c-----------------------------------------------------------------
73:c  Program information echo
74:c-----------------------------------------------------------------
75:c
76:     write(6,7000) sysname
77: 7000 format(
78:    & /1x,'=====================================================',
79:    & /1x,' Fusion reactor tritium inventory evaluation system',
80:    & /1x,'  Version 1.0 on ',a/
81:    & /1x,' 1997 The Japan Atomic Energy Research Institute',
82:    & /1x,'====================================================='/)
83:c
84:c ============================================================
85:c   Initialize variable name table and set their attributes,
86:c   And set default constant values
87:c ============================================================
88:c
89:     lpr = 1
90:     call definevar( errsw, lpr )
91:c
92:c ============================================================
93:c   Set default problem control parameters ......
94:c ============================================================
95:c
96:c   ... maximum calculation time (hour)
97:c
98:     tmax  = 2200d0/3600d0
99:c
100:c   ... number of time steps
101:c
102:     ntimes  = MAXTIMES
103:c
104:c   ... default/current time step width
105:c
106:     delta_t = tmax / ntimes
107:c
108:c   ... cycle number
109:c
```

```
      ncycle  = 79
c
c ==============================================================
c   Read problem control information file
c ==============================================================
c
      call CONTROLINPUT
c
c ==============================================================
c   Solver.
c ==============================================================
c
      LIMIT   = MAXSTORAGE
c
      LX0     = 1
      LXJ     = LX0  + nvar
      LX      = LXJ  + nvar
      LXX     = LX  + nvar
      LX1     = LXX  + nvar
      LXW     = LX1  + nvar
      LF      = LXW  + nvar
      LF1     = LF  + ndfvar
      LAST    = LF1  + ndfvar
c
      if( LAST.gt.LIMIT+1 ) then
        write(6,*) '=== data arae requires ',LAST-1,' double words ',
     &      ' but program limit is ',limit
        stop 999
      end if
c
      call SOLVER( tmax, ntimes, times, nsteps,
     &  ntimezone, delta_t, method, sysname,
     &  jdebug, jdisp,
     &  D(LF), ncycle, nstartc, nbreakc, nbreakp,
     &  dnsgmT, mode, ntimes_r, times_r  )
c
c
      stop
      end
```

● _initprog.f

```
      subroutine initprog( sysname )
c=====================================================================
c     Fusion reactor tritium inventory evaluation system
c---------------------------------------------------------------------
c   system dependent program initialization
c---------------------------------------------------------------------
c $Id: _initprog.f,v 1.3 1997/05/07 02:18:32 sasaki Exp $
c=====================================================================
      character*(*) sysname
c
c---------------------------------------------------------------------
c  For Macintosh (Absoft Fortran77/Mac Fortran II)
c---------------------------------------------------------------------
c
      sysname = 'Macintosh'
c
c---------------------------------------------------------------------
c  Open stdin as NULL File name pops up a file name input menu
c---------------------------------------------------------------------
      open(5, file="", status='OLD' )
c
      return
      end
```

● _fileninp.f

```
      subroutine FILENAMEINPUT
c=====================================================================
c     Fusion reactor tritium inventory evaluation system
c---------------------------------------------------------------------
c Input file or directory/folder names and calculation options.
c---------------------------------------------------------------------
c $Id: _fileninp.f,v 1.8 1998/03/26 05:22:22 sasaki Exp $
c---------------------------------------------------------------------
c=====================================================================
      include '_sizes.h'
      include '_commons1.h'
c
*     character*128 print
*     character*128 result
*     common /FILES/ print, result, response
c*********************************************************************
c*  ... file names
c*
c*   PRINT : standard output ( I/O unit 6 )
c*   RESULT : top directory of result output files (I/O unit 10)
c*   RESPONSE : response file name
c*
c*    In Mac - Directory separator is ":" and put ":" on
c*             the head if the directory path is relative path
c*             ( as opposed to DOS/UNIX convention.)
c*
c*   OUTPUTALL = 1   (output calculation results of all variable
c*                   overriding specification in $OUTPUT)
c*   JDEBUG = 1      (debug output on if non zero)
c*
c*********************************************************************
c
      character*128 cwork
      namelist /FILES/ PRINT, RESULT, RESPONSE, METHOD, JDEBUG,
     &             OUTPUTALL, JDISP
c
```

page 3

```
 37:     PRINT  = ' '
 38:     RESULT = ' '
 39:     RESPONSE = ' '
 40:     METHOD = ' '
 41:     JDEBUG = 0
 42:     JDISP = 0
 43:     OUTPUTALL = 0
 44:c
 45:     write(6,*) '-----------------------------------------------------'
 46:     write(6,*) ' ... input file/folder name and options ...........'
 47:     write(6,*) '-----------------------------------------------------'
 48:c
 49:c----** input file names by namelist **--------------------------------
 50:c
 51:     read(unit=50,nml =FILES)
 52:c
 53:c
 54:     if ( PRINT.ne.' ' ) then
 55:c      LLP    = INDEX(PRINT,' ') - 1
 56:c      if ( LLP.lt.0 ) LLP = LEN(PRINT)
 57:       do LLP = len(PRINT),1,-1
 58:         if (PRINT(LLP:LLP).ne.' ') goto 140
 59:       end do
 60:       LLP =1
 61: 140   continue
 62:
 63:       open( 6, file =PRINT(:LLP), status ='UNKNOWN', err =100,
 64:    &        iostat =IOS )
 65:     end if
 66:c
 67:     if ( RESULT.ne.' ' ) then
 68:c
 69:c ... remove ':' on the head if system is not Macintosh
 70:c
 71:       if( result(1:1).eq.':' .and. sysname.ne.'Macintosh' ) then
 72:         cwork = result(2:)
 73:         result = cwork
 74:         write(6,*) '!!! ":" on the head of RESULT directory name',
 75:    &              ' is removed.'
 76:       end if
 77:c
 78:c      LLR    = INDEX(RESULT,' ') - 1
 79:c      if ( LLR.lt.0 ) LLR = LEN(RESULT)
 80:c
 81:       do LLR = len(RESULT),1,-1
 82:         if (RESULT(LLR:LLR).ne.' ') goto 150
 83:       end do
 84:       LLR = 1
 85: 150   continue
 86:c
 87:c      open( 10, file =RESULT(:LLR), status ='UNKNOWN', form =
 88:c   &        'FORMATTED', err =110, iostat =IOS )
 89:c
 90:     end if
 91:c
 92:     if ( RESPONSE.ne.' ' ) then
 93:       do LLRS = len(RESPONSE),1,-1
 94:         if (RESPONSE(LLRS:LLRS).ne.' ') goto 160
 95:       end do
 96:       LLRS = 1
 97: 160   continue
 98:     else
 99:       RESPONSE = 'response.file'
100:       LLRS = len(RESPONSE)
101:     end if
102:c
103:     if ( PRINT.ne.' ' ) then
104:       write(6,*) ' ... printout filename : <', PRINT(:LLP), '>'
105:     end if
106:     if ( RESULT.ne.' ' ) then
107:       write(6,*) ' ... result output directory : <',RESULT(:LLR),'>'
108:     end if
109:     write(6,*) ' ... response filename : <', RESPONSE(:LLRS), '>'
110:c
111:c ...
112:c
113:     if(METHOD.eq.'EULER' ) then
114:       write(6,*) '=== Problem is solved by explicit Euler method.'
115:       METHOD = 'EULER'
116:     else if( METHOD.eq.' '.or.METHOD.eq.'RK' ) then
117:       write(6,*)
118:    &      '=== Problem is solved by 4th order Runge-Kutta method.'
119:       METHOD = 'RK'
120:     else
121:       write(6,*) 'XXX unsupported solution method. METHOD=<',
122:    &           METHOD,'>'
123:       stop 888
124:     end if
125:c
126:     if(JDEBUG.ne.0) then
127:       write(6,*) '!!! Specified debug option level ',jdebug
128:     end if
129:     if(OUTPUTALL.ne.0) then
130:       write(6,*) '!!! output all results.',
131:    &           ' Specifications in $OUTPUT are ignored.',
132:    &           outputall
133:     end if
134:c
135:     return
136:c
137:c----** error message section **-----------------------------------------
138:c
139: 100 write(6,*) 'xxxx failed to open stdout as named file <',
140:    &      PRINT(:LLP), '> error code = ', IOS
141:     stop 888
142:c
143:c 110 write(6,*) 'xxxx failed to open result file <', RESULT(:LLR),
144:c   &      '> error code = ', IOS
145:c    stop 888
146:c
147:     end
```

● defvar.f

```
 1:     subroutine definevar(ierr,lpr)
 2:C ***********************************************
 3:C   Automatically created.
 4:C   creation date : << Tue May 6 14:21:39 1997 >>
 5:C ***********************************************
 6:C==============================================================
 7:C ... variables & constants definition in varconst.h
 8:C
 9:C
10:C varname(*)  :  name of variables.
11:C vardesc(*)  :  description of variables.
12:C ivarflag(*)  :  type of variables.
13:C    =0 : does not appear in mode equations.
14:C    =1 : dX/dt is specified for this variable.
15:C    =2 : directly or indirectly refered in dX/dt =...
16:C    =3 : not related to dX/dt =... but refered
17:C         in definition of other variables.
18:C    =4 : not related to dX/dt =... and not refered
19:C         in definition of any variables.
20:C
21:C   Variables whose = 1, 2, 3 needs initial value.
22:C
23:C constname(*)  :  name of constants.
24:C constdesc(*)  :  description of constants.
25:C   ( Here "constant" means variables whose values are
26:C     over simulation time. Their values can be changed to
27:C     ppecified values during input phase of program.)
28:C==============================================================
29:     implicit double precision (a-h,o-z)
30:     character*64 srcid
31:     integer ierr,lpr
32:C
33:     include '_sizes.h'
34:     include 'varconst.h'
35:
36:C ... statement function ...
37:     ABX(AA,BB,XX) = AA+ BB*XX
38:C ... ierr (>1:error occured ,=0 > no error)
39:     ierr = 1
40:     srcid = 'Tue May 6 14:21:39 1997'
41:     write(6,*) '=== model translation on:',
42:    &         'Tue May 6 14:21:39 1997'
43:C
44:C --- Variable names & descriptions
45:C
46:C
47:     varname(1) = 'Pfusion(t)'
48:     vardesc(1) = 'fusion power (MW)'
49:     ivarflag(1) = 2
50:C
51:     varname(2) = 'Fuelf_t(t)'
52:     vardesc(2) = 'amount of tritium deliberately fueled to the '
53:    &         //'plasma from fuelers (mole/h)'
54:     ivarflag(2) = 2
55:C
56:     varname(3) = 'Fuel_t(t)'
57:     vardesc(3) = 'total tritium fueled to the plasma (mole/h)'
58:     ivarflag(3) = 2
59:C
60:     varname(4) = 'Fuel_d(t)'
61:     vardesc(4) = 'total deuterium fueled to the plasma (mole/h)'
62:     ivarflag(4) = 2
63:C
64:     varname(5) = 'Fuel_p(t)'
65:     vardesc(5) = 'total protium sources to the plasma (mole/h)'
66:     ivarflag(5) = 2
67:C
68:     varname(6) = 'Burn_t(t)'
69:     vardesc(6) = 'tritium burnup rate (moles/h)'
70:     ivarflag(6) = 2
71:C
72:     varname(7) = 'Burn_d(t)'
73:     vardesc(7) = 'deuterium burnup rate (moles/h)'
74:     ivarflag(7) = 2
75:C
76:     varname(8) = 'Reac_p(t)'
77:     vardesc(8) = 'formation rate of protium from advanced fuel '
78:    &         //'reactions (mole/h)'
79:     ivarflag(8) = 2
80:C
81:     varname(9) = 'Efusion(t)'
82:     vardesc(9) = 'amount of fusion energy deliberately caused by '
83:    &         //'one reaction (MeV)'
84:     ivarflag(9) = -1
85:C
86:     varname(10) = 'Fex_t(t)'
87:     vardesc(10) = 'exhaust rate of tritium from plasma (mole/h)'
88:     ivarflag(10) = 2
89:C
90:     varname(11) = 'Fex_d(t)'
91:     vardesc(11) = 'exhaust rate of deuterium from plasma (mole/h)'
92:     ivarflag(11) = 2
93:C
94:     varname(12) = 'Fex_p(t)'
95:     vardesc(12) = 'exhaust rate of protium from plasma (mole/h)'
96:     ivarflag(12) = 2
97:C
98:     varname(13) = 'Fhe_ex(t)'
99:     vardesc(13) = 'exhaust rate of helium from plasma (mole/h)'
100:    ivarflag(13) = 2
101:C
102:    varname(14) = 'i_t(t)'
103:    vardesc(14) = 'isotopic fraction of exhaust tritium from plasma'
104:    ivarflag(14) = 2
105:C
106:    varname(15) = 'i_d(t)'
107:    vardesc(15) = 'isotopic fraction of exhaust deuterium from '
108:   &         //'plasma'
109:    ivarflag(15) = 2
110:C
111:    varname(16) = 'i_p(t)'
```

page 5

```
      vardesc(16) = 'isotopic fraction of exhaust protium from plasma'
      ivarflag(16) = 2
C
      varname(17) = 'Adiv_t(t)'
      vardesc(17) = 'adsorption of tritium from the edge into '
     &          //'divertor (moles)'
      ivarflag(17) = 2
C
      varname(18) = 'Afw_t(t)'
      vardesc(18) = 'adsorption of tritium from the edge into first '
     &          //'wall (moles)'
      ivarflag(18) = 2
C
      varname(19) = 'd_Adiv_t(t)'
      vardesc(19) = 'rate of adsorption of tritium from the edge '
     &          //'into divertor (mole/h)'
      ivarflag(19) = 2
C
      varname(20) = 'd_Afw_t(t)'
      vardesc(20) = 'rate ofadsorption of tritium from the edge into '
     &          //'first wall (mole/h)'
      ivarflag(20) = 2
C
      varname(21) = 'C_t(t)'
      vardesc(21) = 'tritium codeposited onto plasma facing '
     &          //'components (moles)'
      ivarflag(21) = 1
C
      varname(22) = 'd_C_t(t)'
      vardesc(22) = 'rate of tritium codeposition onto plasma facing '
     &          //'components (mole/h)'
      ivarflag(22) = 2
C
      varname(23) = 'D_t(t)'
      vardesc(23) = 'tritium in tokamak dust (moles)'
      ivarflag(23) = 1
C
      varname(24) = 'd_D_t(t)'
      vardesc(24) = 'rate of change of tritium in tokamak dust '
     &          //'(mole/h)'
      ivarflag(24) = 2
C
      varname(25) = 'Adiv_d(t)'
      vardesc(25) = 'adsorption of deuterium from edge into divertor '
     &          //'(moles)'
      ivarflag(25) = 2
C
      varname(26) = 'Afw_d(t)'
      vardesc(26) = 'adsorption of deuterium from edge into first '
     &          //'wall (moles)'
      ivarflag(26) = 2
C
      varname(27) = 'd_Adiv_d(t)'
      vardesc(27) = 'rate of adsorption of deuterium from edge into '
     &          //'divertor (mole/h)'
      ivarflag(27) = 2
C
      varname(28) = 'd_Afw_d(t)'
      vardesc(28) = 'rate of adsorption of deuterium from edge into '
     &          //'first wall (mole/h)'
      ivarflag(28) = 2
C
      varname(29) = 'C_d(t)'
      vardesc(29) = 'deuterium codeposited onto plasma facing '
     &          //'components (moles)'
      ivarflag(29) = 1
C
      varname(30) = 'd_C_d(t)'
      vardesc(30) = 'rate of deuterium codeposition onto plasma '
     &          //'facing components (mole/h)'
      ivarflag(30) = 2
C
      varname(31) = 'D_d(t)'
      vardesc(31) = 'deuterium in tokamak dust (moles)'
      ivarflag(31) = 1
C
      varname(32) = 'd_D_d(t)'
      vardesc(32) = 'rate of change of deuterium in tokamak dust '
     &          //'(mole/h)'
      ivarflag(32) = 2
C
      varname(33) = 'Adiv_p(t)'
      vardesc(33) = 'adsorption of protium from edge into divertor '
     &          //'(moles)'
      ivarflag(33) = 2
C
      varname(34) = 'Afw_p(t)'
      vardesc(34) = 'adsorption of protium from edge into first wall '
     &          //'(moles)'
      ivarflag(34) = 2
C
      varname(35) = 'd_Adiv_p(t)'
      vardesc(35) = 'rate of adsorption of protium from edge into '
     &          //'divertor (mole/h)'
      ivarflag(35) = 2
C
      varname(36) = 'd_Afw_p(t)'
      vardesc(36) = 'rate of adsorption of protium from edge into '
     &          //'first wall (mole/h)'
      ivarflag(36) = 2
C
      varname(37) = 'C_p(t)'
      vardesc(37) = 'protium codeposited onto plasma facing '
     &          //'components (moles)'
      ivarflag(37) = 1
C
      varname(38) = 'd_C_p(t)'
      vardesc(38) = 'rate of protium codeposition onto plasma facing '
     &          //'components (mole/h)'
      ivarflag(38) = 2
C
      varname(39) = 'D_p(t)'
      vardesc(39) = 'protium in tokamak dust (moles)'
      ivarflag(39) = 1
```

page 6

```
226:C
227:     varname(40) = 'd_D_p(t)'
228:     vardesc(40) = 'rate of change of protium in tokamak dust '
229:    &          //'(mole/h)'
230:     ivarflag(40) = 2
231:C
232:     varname(41) = 'Tpfc_(t)'
233:     vardesc(41) = 'total tritium in plasma facing componenta (moles)'
234:     ivarflag(41) = 4
235:C
236:     varname(42) = 'Tpfc_m(t)'
237:     vardesc(42) = 'mobilizable tritium in plasma facing components '
238:    &          //'(moles)'
239:     ivarflag(42) = 4
240:C
241:     varname(43) = 'Dpfc_(t)'
242:     vardesc(43) = 'total deuterium in plasma facing components '
243:    &          //'(moles)'
244:     ivarflag(43) = 4
245:C
246:     varname(44) = 'Ppfc_(t)'
247:     vardesc(44) = 'total protium in plasma facing components (moles)'
248:     ivarflag(44) = 4
249:C
250:     varname(45) = 'Tpv(t)'
251:     vardesc(45) = 'tritium inventory per pumps (moles)'
252:     ivarflag(45) = 1
253:C
254:     varname(46) = 'Tp_reg(t)'
255:     vardesc(46) = 'tritium inventory per pump (batch operation)  '
256:    &          //'at regeneration time (moles)'
257:     ivarflag(46) = -1
258:C
259:     varname(47) = 'Tp_tot(t)'
260:     vardesc(47) = 'total tritium inventory on all pumps'
261:     ivarflag(47) = 3
262:C
263:     varname(48) = 'Tp_tot_m(t)'
264:     vardesc(48) = 'total mobilizable tritium inventory on all '
265:    &          //'pumps (moles)'
266:     ivarflag(48) = 4
267:C
268:     varname(49) = 'Rt_rem(t)'
269:     vardesc(49) = 'removal rate of tritium from continuos '
270:    &          //'regeneration cryopump (mole/h)'
271:     ivarflag(49) = 2
272:C
273:     varname(50) = 'Dpv(t)'
274:     vardesc(50) = 'Deuterium inventory per pumps (moles).'
275:     ivarflag(50) = 1
276:C
277:     varname(51) = 'Rd_rem(t)'
278:     vardesc(51) = 'removal rate of deuterium from continuos '
279:    &          //'regeneration cryopump (mole/h)'
280:     ivarflag(51) = 2
281:C
282:     varname(52) = 'Dp_reg(t)'
283:     vardesc(52) = 'tritium inventory per pump (batch operation)  '
284:    &          //'at regeneration time (moles)'
285:     ivarflag(52) = -1
286:C
287:     varname(53) = 'Dp_tot(t)'
288:     vardesc(53) = 'total deuterium inventory on all pumps (moles).'
289:     ivarflag(53) = 4
290:C
291:     varname(54) = 'Rp_rem(t)'
292:     vardesc(54) = 'removal rate of protium from continuos '
293:    &          //'regeneration cryopump (mole/h).'
294:     ivarflag(54) = 2
295:C
296:     varname(55) = 'Pp_reg(t)'
297:     vardesc(55) = 'protium inventory per pump (batch operation)  '
298:    &          //'at regeneration time (moles).'
299:     ivarflag(55) = -1
300:C
301:     varname(56) = 'Pp_tot(t)'
302:     vardesc(56) = 'total protium inventory on all pumps (moles).'
303:     ivarflag(56) = 4
304:C
305:     varname(57) = 'Ppv(t)'
306:     vardesc(57) = 'protium inventory per pumps (moles).'
307:     ivarflag(57) = 1
308:C
309:     varname(58) = 'Rhe_rem(t)'
310:     vardesc(58) = 'removal rate of helium from continuos '
311:    &          //'regeneration cryopump (mole/h)'
312:     ivarflag(58) = 2
313:C
314:     varname(59) = 'Hepv(t)'
315:     vardesc(59) = 'helium inventory per pumps (moles).'
316:     ivarflag(59) = 1
317:C
318:     varname(60) = 'Hep_reg(t)'
319:     vardesc(60) = 'helium inventory per pump (batch operation) at '
320:    &          //'regeneration time (moles).'
321:     ivarflag(60) = -1
322:C
323:     varname(61) = 'Hep_tot(t)'
324:     vardesc(61) = 'total deuterium inventory on all pumps (moles)'
325:     ivarflag(61) = 4
326:C
327:     varname(62) = 'Fv_out(t)'
328:     vardesc(62) = 'total flow from vacuum pumps to fuel clean-up '
329:    &          //'unit (mole/h).'
330:     ivarflag(62) = 2
331:C
332:     varname(63) = 'Fpuri_in(t)'
333:     vardesc(63) = 'total flow from vacuum pumps to purification '
334:    &          //'(mole/h)'
335:     ivarflag(63) = 4
336:C
337:     varname(64) = 'it_(t)'
338:     vardesc(64) = 'isotopic fraction of tritium on vacuum system '
339:    &          //'cryopumps'
```

page 7

```
340:     ivarflag(64) = 2
341:C
342:     varname(65) = 'id_(t)'
343:     vardesc(65) = 'isotopic fraction of deuterium on vacuum system '
344:    &         //'cryopumps'
345:     ivarflag(65) = 2
346:C
347:     varname(66) = 'ip_(t)'
348:     vardesc(66) = 'isotopic fraction of protium on vacuum system '
349:    &         //'cryopumps'
350:     ivarflag(66) = 2
351:C
352:     varname(67) = 'xnhydro_(t)'
353:     vardesc(67) = 'fractin of non-hydrogen containing impurities '
354:    &         //'in adjusted flow to purification'
355:     ivarflag(67) = 2
356:C
357:     varname(68) = 'xh_imp(t)'
358:     vardesc(68) = 'fraction of hydrogen containing impurities in '
359:    &         //'adjusted flow to purification'
360:     ivarflag(68) = 2
361:C
362:     varname(69) = 'Fadj_puri(t)'
363:     vardesc(69) = 'adjusted flow rate to purification considering '
364:    &         //'impurities (mole/h)'
365:     ivarflag(69) = 2
366:C
367:     varname(70) = 'Fhydro_pure(t)'
368:     vardesc(70) = 'flow of pure atoms to purification (mole/h)'
369:     ivarflag(70) = -1
370:C
371:     varname(71) = 'Fhydro_imp(t)'
372:     vardesc(71) = 'flow of hydrogen containing impurities to '
373:    &         //'purification (mole/h)'
374:     ivarflag(71) = -1
375:C
376:     varname(72) = 'Fnhydro_imp(t)'
377:     vardesc(72) = 'flow of non-hydrogen containing impurities to '
378:    &         //'purification (mole/h)'
379:     ivarflag(72) = 4
380:C
381:     varname(73) = 'Ft_pure(t)'
382:     vardesc(73) = 'flow of pure tritium atoms to purification '
383:    &         //'(mole/h)'
384:     ivarflag(73) = 2
385:C
386:     varname(74) = 'Fd_pure(t)'
387:     vardesc(74) = 'flow of pure deuterium atoms to purification '
388:    &         //'(mole/h)'
389:     ivarflag(74) = 2
390:C
391:     varname(75) = 'Fp_pure(t)'
392:     vardesc(75) = 'flow of pure protium atoms to purification '
393:    &         //'(mole/h)'
394:     ivarflag(75) = 2
395:C
396:     varname(76) = 'Fcq4_imp(t)'
397:     vardesc(76) = 'flow of methane to purification (mole/h)'
398:     ivarflag(76) = 2
399:C
400:     varname(77) = 'Fnq3_imp(t)'
401:     vardesc(77) = 'flow of ammonia to purification (mole/h)'
402:     ivarflag(77) = 2
403:C
404:     varname(78) = 'Fq2o_imp(t)'
405:     vardesc(78) = 'flow of water to purification (mole/h)'
406:     ivarflag(78) = 2
407:C
408:     varname(79) = 'Fhydro_imp_w(t)'
409:     vardesc(79) = 'constituents of hydrogen flow containing '
410:    &         //'impurities (mole/h)'
411:     ivarflag(79) = 2
412:C
413:     varname(80) = 'f_hydro_imp_w(t)'
414:     vardesc(80) = 'weighting fraction of constituents of hydrogen '
415:    &         //'flow containing impurities'
416:     ivarflag(80) = 2
417:C
418:     varname(81) = 'Nimpt(t)'
419:     vardesc(81) = 'total impurities on purification unit (moles)'
420:     ivarflag(81) = 1
421:C
422:     varname(82) = 'Nhydro_imp(t)'
423:     vardesc(82) = 'total hydrogen in hydrogen-containing '
424:    &         //'impurities on purification unit (moles)'
425:     ivarflag(82) = 2
426:C
427:     varname(83) = 'Nnhydro_imp(t)'
428:     vardesc(83) = 'total non-hydro impurities on purification unit '
429:    &         //'(moles)'
430:     ivarflag(83) = 1
431:C
432:     varname(84) = 'Ncq4_imp(t)'
433:     vardesc(84) = 'methane on purification unit (moles)'
434:     ivarflag(84) = 1
435:C
436:     varname(85) = 'Nnq3_imp(t)'
437:     vardesc(85) = 'ammonia on purification unit (moles)'
438:     ivarflag(85) = 1
439:C
440:     varname(86) = 'Nq2o_imp(t)'
441:     vardesc(86) = 'water on purification unit (moles)'
442:     ivarflag(86) = 1
443:C
444:     varname(87) = 'Timp_(t)'
445:     vardesc(87) = 'total tritium on purification unit (moles)'
446:     ivarflag(87) = 1
447:C
448:     varname(88) = 'Dimp_(t)'
449:     vardesc(88) = 'total deuterium on purification unit (moles)'
450:     ivarflag(88) = 1
451:C
452:     varname(89) = 'Pimp_(t)'
453:     vardesc(89) = 'total protium on purification unit (moles)'
```

```
454:     ivarflag(89) = 1
455:C
456:     varname(90) = 'Fh2_puri_in(t)'
457:     vardesc(90) = 'flow of pure hydrogen molecules to purification '
458:    &          //'(mole/h)'
459:     ivarflag(90) = 4
460:C
461:     varname(91) = 'Ft2_sub_in(t)'
462:     vardesc(91) = 'flow of pure tritium molecules to standby unit '
463:    &          //'(mole/h)'
464:     ivarflag(91) = 2
465:C
466:     varname(92) = 'Fd2_sub_in(t)'
467:     vardesc(92) = 'flow of pure deuterium molecules to standby '
468:    &          //'unit (mole/h)'
469:     ivarflag(92) = 2
470:C
471:     varname(93) = 'Fp2_sub_in(t)'
472:     vardesc(93) = 'flow of pure protium molecules to standby unit '
473:    &          //'(mole/h)'
474:     ivarflag(93) = 2
475:C
476:     varname(94) = 'Fh2_sub_in(t)'
477:     vardesc(94) = 'flow of pure fydrogen molecules to standby unit '
478:    &          //'(mole/h)'
479:     ivarflag(94) = 2
480:C
481:     varname(95) = 'Ns(t)'
482:     vardesc(95) = 'number of unfilled adsorption sites for '
483:    &          //'hydrogen on standby uint (moles)'
484:     ivarflag(95) = 1
485:C
486:     varname(96) = 'xt_sub(t)'
487:     vardesc(96) = 'mole fraction of tritium entering standby unit'
488:     ivarflag(96) = 2
489:C
490:     varname(97) = 'xd_sub(t)'
491:     vardesc(97) = 'mole fraction of deuterium entering standby unit'
492:     ivarflag(97) = 2
493:C
494:     varname(98) = 'xp_sub(t)'
495:     vardesc(98) = 'mole fraction of protium entering standby unit'
496:     ivarflag(98) = 2
497:C
498:     varname(99) = 'T2_sub(t)'
499:     vardesc(99) = 'tritium molecules on standby unit (moles)'
500:     ivarflag(99) = 1
501:C
502:     varname(100) = 'D2_sub(t)'
503:     vardesc(100) = 'deuterium molecules on standby unit (moles)'
504:     ivarflag(100) = 1
505:C
506:     varname(101) = 'P2_sub(t)'
507:     vardesc(101) = 'protium molecules on standby unit (moles)'
508:     ivarflag(101) = 1
509:C
510:     varname(102) = 'Nhydro_sub(t)'
511:     vardesc(102) = 'total hydrogen on standby unit (moles)'
512:     ivarflag(102) = 4
513:C
514:     varname(103) = 'Ft_sub_out(t)'
515:     vardesc(103) = 'flow of pure tritium atoms out of standby unit '
516:    &          //'(mole/h)'
517:     ivarflag(103) = 2
518:C
519:     varname(104) = 'Fd_sub_out(t)'
520:     vardesc(104) = 'flow of pure deuterium atoms out of standby '
521:    &          //'unit (mole/h)'
522:     ivarflag(104) = 2
523:C
524:     varname(105) = 'Fp_sub_out(t)'
525:     vardesc(105) = 'flow of pure protium atoms out of standby unit '
526:    &          //'(mole/h)'
527:     ivarflag(105) = 2
528:C
529:     varname(106) = 'Fh_sub_out(t)'
530:     vardesc(106) = 'flow of pure hydrogen atoms out of standby unit '
531:    &          //'(mole/h)'
532:     ivarflag(106) = 4
533:C
534:     varname(107) = 'Treg_out(t)'
535:     vardesc(107) = 'tritium on regenerating molecular sieve beds '
536:    &          //'(moles)'
537:     ivarflag(107) = 3
538:C
539:     varname(108) = 'Ecq4_reg(t)'
540:     vardesc(108) = 'evolution of methane from molecular sieve '
541:    &          //'during regeneration (mole/h)'
542:     ivarflag(108) = 2
543:C
544:     varname(109) = 'Enq3_reg(t)'
545:     vardesc(109) = 'evolution of ammonia from molecular sieve '
546:    &          //'during regeneration (mole/h)'
547:     ivarflag(109) = 2
548:C
549:     varname(110) = 'Eq2o_reg(t)'
550:     vardesc(110) = 'evolution of water from molecular sieve during '
551:    &          //'regeneration (mole/h)'
552:     ivarflag(110) = 2
553:C
554:     varname(111) = 'Ncq4_reg(t)'
555:     vardesc(111) = 'amount of methane on purification bed at '
556:    &          //'regeneration (moles)'
557:     ivarflag(111) = 2
558:C
559:     varname(112) = 'Nnq3_reg(t)'
560:     vardesc(112) = 'amount of ammonia on purification bed at '
561:    &          //'regeneration (moles)'
562:     ivarflag(112) = 2
563:C
564:     varname(113) = 'Nq2o_reg(t)'
565:     vardesc(113) = 'amount of water on purification bed at '
566:    &          //'regeneration (moles)'
567:     ivarflag(113) = 2
```

page 9

```
C
      varname(114) = 'Fhydro_pmem(t)'
      vardesc(114) = 'flow of hydrogen to palladium membrane reactor '
     &          //'(mole/h)'
      ivarflag(114) = 2
C
      varname(115) = 'Tpmem(t)'
      vardesc(115) = 'tritium on palladium membrane reactor (moles)'
      ivarflag(115) = 1
C
      varname(116) = 'Dpmem(t)'
      vardesc(116) = 'deuterium on palladium membrane reactor (moles)'
      ivarflag(116) = 1
C
      varname(117) = 'Ppmem(t)'
      vardesc(117) = 'protium on palladium membrane reactor (moles)'
      ivarflag(117) = 1
C
      varname(118) = 'it_endj(t)'
      vardesc(118) = 'isotopic fraction of tritium on molecular sieve '
     &          //'at end of purification'
      ivarflag(118) = 2
C
      varname(119) = 'id_endj(t)'
      vardesc(119) = 'isotopic fraction of deuterium on molecular '
     &          //'sieve at end of purification'
      ivarflag(119) = 2
C
      varname(120) = 'ip_endj(t)'
      vardesc(120) = 'isotopic fraction of protium on molecular sieve '
     &          //'at end of purification'
      ivarflag(120) = 2
C
      varname(121) = 'Ft_puri_out(t)'
      vardesc(121) = 'flow of tritium to isotope separarion system '
     &          //'(mole/h)'
      ivarflag(121) = 2
C
      varname(122) = 'Fd_puri_out(t)'
      vardesc(122) = 'flow of deuterium to isotope separation system '
     &          //'(mole/h)'
      ivarflag(122) = 2
C
      varname(123) = 'Fp_puri_out(t)'
      vardesc(123) = 'flow of protium to isotope separation system '
     &          //'(mole/h)'
      ivarflag(123) = 2
C
      varname(124) = 'Fpuri_out(t)'
      vardesc(124) = 'total flow to isotope separation system (mole/h)'
      ivarflag(124) = 2
C
      varname(125) = 'Xiss_in_t(t)'
      vardesc(125) = 'mole fraction of tritium in stream from fuel '
     &          //'clean-up unit to isotope separation system'
      ivarflag(125) = 2
C
      varname(126) = 'Xiss_in_d(t)'
      vardesc(126) = 'mole fraction of deuterium in stream from fuel '
     &          //'clean-up unit to isotope separation system'
      ivarflag(126) = 2
C
      varname(127) = 'Xiss_in_p(t)'
      vardesc(127) = 'mole fraction of protium in stream from fuel '
     &          //'clean-up unit to isotope separation system'
      ivarflag(127) = 4
C
      varname(128) = 'Fpure_(t)'
      vardesc(128) = 'total flow of pure tritium, deuterium, protium '
     &          //'and helium atoms to purification (mole/h)'
      ivarflag(128) = 4
C
      varname(129) = 'Fimp_(t)'
      vardesc(129) = 'total flowa of impurities to purification '
     &          //'(mole/h)'
      ivarflag(129) = 2
C
      varname(130) = 'Timp_m(t)'
      vardesc(130) = 'total mobilizable tritium inventory on '
     &          //'purification unit (moles)'
      ivarflag(130) = 4
C
      varname(131) = 'Treg_out_m(t)'
      vardesc(131) = 'mobilizable tritium on regenerating molecular '
     &          //'sieve beds (moles)'
      ivarflag(131) = 4
C
      varname(132) = 'Fiss_in(t)'
      vardesc(132) = 'total flow to isotope separation system (mole/h)'
      ivarflag(132) = 2
C
      varname(133) = 'Fiss_nb(t)'
      vardesc(133) = 'flow from neutral beam to isotope separation '
     &          //'system (mole/h)'
      ivarflag(133) = 4
C
      varname(134) = 'Fiss_in_t(t)'
      vardesc(134) = 'flow of tritium to isotope separation system '
     &          //'(mole/h)'
      ivarflag(134) = 2
C
      varname(135) = 'Xiss_t(t)'
      vardesc(135) = 'mole fraction of tritium in stream from fuel '
     &          //'clean-up unit'
      ivarflag(135) = 3
C
      varname(136) = 'Xnb_t(t)'
      vardesc(136) = 'mole fraction of tritium in stream from neutral '
     &          //'beam cryopump'
      ivarflag(136) = 4
C
      varname(137) = 'Xinj_t(t)'
      vardesc(137) = 'mole fraction of tritium in pellet injector '
     &          //'exhaust stream'
```

page 10

```
682:     ivarflag(137) = 3
683:C
684:     varname(138) = 'Fiss_in_d(t)'
685:     vardesc(138) = 'flow of deuterium to isotope separation system '
686:    &         //'(mole/h)'
687:     ivarflag(138) = 2
688:C
689:     varname(139) = 'Xiss_d(t)'
690:     vardesc(139) = 'mole fraction of deuterium in stream from fuel '
691:    &         //'clean-up unit'
692:     ivarflag(139) = 3
693:C
694:     varname(140) = 'Xnb_d(t)'
695:     vardesc(140) = 'mole fraction of deuterium in stream from '
696:    &         //'neutral beam cryopump fuel clean-up unit'
697:     ivarflag(140) = 4
698:C
699:     varname(141) = 'Xinj_d(t)'
700:     vardesc(141) = 'mole fraction of deuterium in pellet injector '
701:    &         //'exhaust stream'
702:     ivarflag(141) = 3
703:C
704:     varname(142) = 'Fiss_in_p(t)'
705:     vardesc(142) = 'flow of protium to isotope separation system '
706:    &         //'(mole/h)'
707:     ivarflag(142) = 4
708:C
709:     varname(143) = 'Niss_(t)'
710:     vardesc(143) = 'total amount in isotope separation system (moles)'
711:     ivarflag(143) = 1
712:C
713:     varname(144) = 'Tiss_(t)'
714:     vardesc(144) = 'amount of tritium in isotope separation system '
715:    &         //'(moles)'
716:     ivarflag(144) = 1
717:C
718:     varname(145) = 'Tiss_m(t)'
719:     vardesc(145) = 'amount of mobilizable tritium in isotope '
720:    &         //'saparation system (moles)'
721:     ivarflag(145) = 4
722:C
723:     varname(146) = 'Diss_(t)'
724:     vardesc(146) = 'amount of deuterium in isotope separation '
725:    &         //'system (moles)'
726:     ivarflag(146) = 1
727:C
728:     varname(147) = 'Piss_(t)'
729:     vardesc(147) = 'amount of protium in isotope separation system '
730:    &         //'(moles)'
731:     ivarflag(147) = 4
732:C
733:     varname(148) = 'Xiss_p(t)'
734:     vardesc(148) = 'mole fraction of protium in isotope separation '
735:    &         //'system'
736:     ivarflag(148) = 3
737:C
738:     varname(149) = 'Fiss_top_(t)'
739:     vardesc(149) = 'flow rate from top of isotope separation system '
740:    &         //'(mole/h)'
741:     ivarflag(149) = 2
742:C
743:     varname(150) = 'Fiss_mid_(t)'
744:     vardesc(150) = 'flow rate from middle of isotope separation '
745:    &         //'system (mole/h)'
746:     ivarflag(150) = 2
747:C
748:     varname(151) = 'Fiss_bot_(t)'
749:     vardesc(151) = 'flow rate from bottom of isotope separation '
750:    &         //'system (mole/h)'
751:     ivarflag(151) = 2
752:C
753:     varname(152) = 'Fiss_inj(t)'
754:     vardesc(152) = 'flow directed to isotope separation system from '
755:    &         //'pellet injector (mole/h)'
756:     ivarflag(152) = 4
757:C
758:     varname(153) = 'Fplas_nb_d(t)'
759:     vardesc(153) = 'flow of deuterium to plasma passing neutral '
760:    &         //'beam (mole/h)'
761:     ivarflag(153) = 2
762:C
763:     varname(154) = 'Fin_st_d(t)'
764:     vardesc(154) = 'flow from deuterium rich storage (mole/h)'
765:     ivarflag(154) = 2
766:C
767:     varname(155) = 'Fin_rec_(t)'
768:     vardesc(155) = 'recycle flow rate to neutral beam from neutral '
769:    &         //'beam cryopump (mole/h)'
770:     ivarflag(155) = 4
771:C
772:     varname(156) = 'Fin_iss_mid(t)'
773:     vardesc(156) = 'flow rate from middle of isotope separation '
774:    &         //'system (mole/h)'
775:     ivarflag(156) = -1
776:C
777:     varname(157) = 'Xcryo_d(t)'
778:     vardesc(157) = 'mole fraction of deuterium on neutral beam '
779:    &         //'cryopump'
780:     ivarflag(157) = -1
781:C
782:     varname(158) = 'W_nb(t)'
783:     vardesc(158) = 'required neutral beam power to be delivered to '
784:    &         //'the plasma (MW)'
785:     ivarflag(158) = 2
786:C
787:     varname(159) = 'Ncryo_(t)'
788:     vardesc(159) = 'total amount on neutral beam cryopump (moles)'
789:     ivarflag(159) = 1
790:C
791:     varname(160) = 'Fbk_nb_(t)'
792:     vardesc(160) = 'total backflow from plasma into neutral beam '
793:    &         //'(mole/h)'
794:     ivarflag(160) = 2
795:C
```

page 11

```
796:     varname(161) = 'Dcryo_(t)'
797:     vardesc(161) = 'deuterium on neutral beam cryopump (moles)'
798:     ivarflag(161) = -1
799:C
800:     varname(162) = 'Dnb_reg_(t)'
801:     vardesc(162) = 'amount of deuterium on the neutral beam batch '
802:    &         //'regeneration (moles)'
803:     ivarflag(162) = -1
804:C
805:     varname(163) = 'Tnb_cryo(t)'
806:     vardesc(163) = 'tritium on neutral beam cryopump (moles)'
807:     ivarflag(163) = 1
808:C
809:     varname(164) = 'Fbk_nb_t(t)'
810:     vardesc(164) = 'backflow of deuterium from plasma into neutral '
811:    &         //'beam (mole/h)'
812:     ivarflag(164) = 2
813:C
814:     varname(165) = 'Rnb_reg_t(t)'
815:     vardesc(165) = 'regeneration rate of tritium from neutral beam '
816:    &         //'cryopump (mole/h)'
817:     ivarflag(165) = 3
818:C
819:     varname(166) = 'tnb_'
820:     vardesc(166) = 'time during pumping cycle for neutral beam '
821:    &         //'pumps (h)'
822:     ivarflag(166) = -1
823:C
824:     varname(167) = 'Tnb_reg_(t)'
825:     vardesc(167) = 'total amount of tritium on the neutral beam '
826:    &         //'batch regeneration cryopump (moles)'
827:     ivarflag(167) = -1
828:C
829:     varname(168) = 'Tnb_cryo_tot(t)'
830:     vardesc(168) = 'total tritium on neutral beam cryopumps (moles)'
831:     ivarflag(168) = -1
832:C
833:     varname(169) = 'Tnb_cryo_mob(t)'
834:     vardesc(169) = 'total mobilizable tritium on neutral beam '
835:    &         //'cryopump (moles)'
836:     ivarflag(169) = 4
837:C
838:     varname(170) = 'Dnb_cryo(t)'
839:     vardesc(170) = 'deuterium on neutral beam cryopump (moles)'
840:     ivarflag(170) = 1
841:C
842:     varname(171) = 'Fbk_nb_d(t)'
843:     vardesc(171) = 'backflow of deuterium from plasma into neutral '
844:    &         //'beam (mole/h)'
845:     ivarflag(171) = 2
846:C
847:     varname(172) = 'Rnb_reg_d(t)'
848:     vardesc(172) = 'regeneration rate of deuterium from neutral '
849:    &         //'beam cryopump (mole/h)'
850:     ivarflag(172) = 2
851:C
852:     varname(173) = 'Dnb_cryo_tot(t)'
853:     vardesc(173) = 'total deuterium on neutral beam cryopumps (moles)'
854:     ivarflag(173) = -1
855:C
856:     varname(174) = 'Xnb_cryo_t(t)'
857:     vardesc(174) = 'mole fraction of tritium on neutral beam cryopump'
858:     ivarflag(174) = 3
859:C
860:     varname(175) = 'Xnb_cryo_d(t)'
861:     vardesc(175) = 'mole fraction of deuterium on neutral beam '
862:    &         //'cryopump'
863:     ivarflag(175) = 3
864:C
865:     varname(176) = 'Xnb_cryo_p(t)'
866:     vardesc(176) = 'mole fraction of protium on neutral beam cryopump'
867:     ivarflag(176) = 3
868:C
869:     varname(177) = 'Pnb_cryo(t)'
870:     vardesc(177) = 'protium on neutral beam cryopump (moles)'
871:     ivarflag(177) = 3
872:C
873:     varname(178) = 'Tnb_wall(t)'
874:     vardesc(178) = 'tritium on neutral beam walls (moles)'
875:     ivarflag(178) = 1
876:C
877:     varname(179) = 'Tnb_wall_mob(t)'
878:     vardesc(179) = 'mobilizable tritium on neutral beam walls (moles)'
879:     ivarflag(179) = 4
880:C
881:     varname(180) = 'Tnb_tot(t)'
882:     vardesc(180) = 'total tritium in neutral beam (moles)'
883:     ivarflag(180) = -1
884:C
885:     varname(181) = 'Tnb_tot_mob(t)'
886:     vardesc(181) = 'total mobilizable tritium in nueutral beam '
887:    &         //'(moles)'
888:     ivarflag(181) = -1
889:C
890:     varname(182) = 'Fnb_cryo(t)'
891:     vardesc(182) = 'flow rate from neutral beam cryopump (mole/h)'
892:     ivarflag(182) = 4
893:C
894:     varname(183) = 'Crec(t)'
895:     vardesc(183) = 'fraction of gas from neutral beam cryopump '
896:    &         //'directed to the isotope separation system'
897:     ivarflag(183) = 4
898:C
899:     varname(184) = 'Fplas_nb_(t)'
900:     vardesc(184) = 'total flow rate to plasma from neutral beam '
901:    &         //'(mole/h)'
902:     ivarflag(184) = 3
903:C
904:     varname(185) = 'Fplas_nb_t(t)'
905:     vardesc(185) = 'flow of tritium to plasma through neutral beam '
906:    &         //'(mole/h)'
907:     ivarflag(185) = 2
908:C
909:     varname(186) = 'Fplas_nb_p(t)'
```

```
910:     vardesc(186) = 'flow of protium to plasma through neutral beam '
911:    &          //'(mole/h)'
912:     ivarflag(186) = 4
913:C
914:     varname(187) = 'ncycle_p(t)'
915:     vardesc(187) = 'number of plasma cycles'
916:     ivarflag(187) = -1
917:C
918:     varname(188) = 'Nnb_cryo(t)'
919:     vardesc(188) = 'total amount on neutral beam cryopump (moles)'
920:     ivarflag(188) = -1
921:C
922:     varname(189) = 'Rnb_reg_(t)'
923:     vardesc(189) = 'total regeneration rate from neutral beam '
924:    &          //'cryopump (mole/h)'
925:     ivarflag(189) = 4
926:C
927:     varname(190) = 'Rnb_reg_p(t)'
928:     vardesc(190) = 'regeneration rate of protium from neutral beam '
929:    &          //'cryopump (mole/h)'
930:     ivarflag(190) = 3
931:C
932:     varname(191) = 'Fgas_t(t)'
933:     vardesc(191) = 'flow of tritium to plasma through gas puffer '
934:    &          //'(mole/h)'
935:     ivarflag(191) = 2
936:C
937:     varname(192) = 'Fgas_in_tri(t)'
938:     vardesc(192) = 'flow from tritium rich storage to gas puffer '
939:    &          //'(mole/h)'
940:     ivarflag(192) = 2
941:C
942:     varname(193) = 'Fgas_in_deu(t)'
943:     vardesc(193) = 'flow from deuterium rich storage to gas puffer '
944:    &          //'(mole/h)'
945:     ivarflag(193) = 2
946:C
947:     varname(194) = 'Fgas_d(t)'
948:     vardesc(194) = 'flow of deuterium to plasma (mole/h)'
949:     ivarflag(194) = 2
950:C
951:     varname(195) = 'Ngas_tot(t)'
952:     vardesc(195) = 'total amount in gas puffer (moles)'
953:     ivarflag(195) = 1
954:C
955:     varname(196) = 'Fgas_tot(t)'
956:     vardesc(196) = 'total flow rate to plasma from gas puffer '
957:    &          //'(mole/h)'
958:     ivarflag(196) = 2
959:C
960:     varname(197) = 'Tgas_(t)'
961:     vardesc(197) = 'amount of tritium in gas puffer (moles)'
962:     ivarflag(197) = 1
963:C
964:     varname(198) = 'Dgas_(t)'
965:     vardesc(198) = 'amount of deuterium in gas puffer (moles)'
966:     ivarflag(198) = 1
967:C
968:     varname(199) = 'Xgas_t(t)'
969:     vardesc(199) = 'mole fraction of tritium in gas puffer exit '
970:    &          //'stream'
971:     ivarflag(199) = 2
972:C
973:     varname(200) = 'Xgas_d(t)'
974:     vardesc(200) = 'mole fraction of deuterium in gas puffer exit '
975:    &          //'stream'
976:     ivarflag(200) = 2
977:C
978:     varname(201) = 'Xgas_p(t)'
979:     vardesc(201) = 'mole fraction of protium in gas puffer exit '
980:    &          //'stream'
981:     ivarflag(201) = 3
982:C
983:     varname(202) = 'Pgas_(t)'
984:     vardesc(202) = 'amount of protium in gas puffer (moles)'
985:     ivarflag(202) = 4
986:C
987:     varname(203) = 'Fpuff_plas(t)'
988:     vardesc(203) = 'total flow to plasma from gas puffer and pellet '
989:    &          //'injector propellant (mole/h)'
990:     ivarflag(203) = 4
991:C
992:     varname(204) = 'Tgas_mob(t)'
993:     vardesc(204) = 'mobilizable tritium in gas puffer (moles)'
994:     ivarflag(204) = 4
995:C
996:     varname(205) = 'Fplas_in_t(t)'
997:     vardesc(205) = 'flow of tritium to plasma through pellet '
998:    &          //'injector (mole/h)'
999:     ivarflag(205) = 2
1000:C
1001:     varname(206) = 'Fplas_in_d(t)'
1002:     vardesc(206) = 'flow of deuterium to plasma through pellet '
1003:    &          //'injector (mole/h)'
1004:     ivarflag(206) = 2
1005:C
1006:     varname(207) = 'Fplas_in_p(t)'
1007:     vardesc(207) = 'flow of protium to plasma through pellet '
1008:    &          //'injector (mole/h)'
1009:     ivarflag(207) = 2
1010:C
1011:     varname(208) = 'Fsto_out_t(t)'
1012:     vardesc(208) = 'flow from tritium rich storage to pellet '
1013:    &          //'injector (mole/h)'
1014:     ivarflag(208) = 2
1015:C
1016:     varname(209) = 'Fsto_out_d(t)'
1017:     vardesc(209) = 'flow from deuterium rich storage to pellet '
1018:    &          //'injector (mole/h)'
1019:     ivarflag(209) = 2
1020:C
1021:     varname(210) = 'Fsto_out_p(t)'
1022:     vardesc(210) = 'flow from protium rich storage to pellet '
1023:    &          //'injector (mole/h)'
```

page 13

```
1024:     ivarflag(210) = 2
1025:C
1026:     varname(211) = 'Next_12(t)'
1027:     vardesc(211) = 'total moles (of atoms) in pellet injector '
1028:    &          //'extruder 1 or 2 (moles)'
1029:     ivarflag(211) = 1
1030:C
1031:     varname(212) = 'Text_12(t)'
1032:     vardesc(212) = 'tritium in pellet injector extruder 1 or 2 '
1033:    &          //'(moles)'
1034:     ivarflag(212) = 1
1035:C
1036:     varname(213) = 'Dext_12(t)'
1037:     vardesc(213) = 'deuterium in pellet injector extruder 1 or 2 '
1038:    &          //'(moles)'
1039:     ivarflag(213) = 1
1040:C
1041:     varname(214) = 'Pext_12(t)'
1042:     vardesc(214) = 'protium in pellet injector extruder 1 or 2 '
1043:    &          //'(moles)'
1044:     ivarflag(214) = 3
1045:C
1046:     varname(215) = 'Xext_12_t(t)'
1047:     vardesc(215) = 'mole fraction of tritium in pellet injector '
1048:    &          //'extruder 1 or 2'
1049:     ivarflag(215) = 3
1050:C
1051:     varname(216) = 'Xext_12_d(t)'
1052:     vardesc(216) = 'mole fraction of deuterium in pellet injector '
1053:    &          //'extruder 1 or 2'
1054:     ivarflag(216) = 3
1055:C
1056:     varname(217) = 'Xext_12_p(t)'
1057:     vardesc(217) = 'mole fraction of protium in pellet injector '
1058:    &          //'extruder 1 or 2'
1059:     ivarflag(217) = 3
1060:C
1061:     varname(218) = 'Next_tot(t)'
1062:     vardesc(218) = 'total in pellet injector extruders (moles)'
1063:     ivarflag(218) = 4
1064:C
1065:     varname(219) = 'Text_tot(t)'
1066:     vardesc(219) = 'total tritium in pellet injector extruders '
1067:    &          //'(moles)'
1068:     ivarflag(219) = 3
1069:C
1070:     varname(220) = 'Dext_tot(t)'
1071:     vardesc(220) = 'total deuterium in pellet injector extruders '
1072:    &          //'(moles)'
1073:     ivarflag(220) = 4
1074:C
1075:     varname(221) = 'Pext_tot(t)'
1076:     vardesc(221) = 'total protium in pellet injector extruders '
1077:    &          //'(moles)'
1078:     ivarflag(221) = 4
1079:C
1080:     varname(222) = 'Text_m(t)'
1081:     vardesc(222) = 'total mobilizable tritium in pellet injectors '
1082:    &          //'(moles)'
1083:     ivarflag(222) = 4
1084:C
1085:     varname(223) = 'Vinj_t(t)'
1086:     vardesc(223) = 'volume of tritium in pellet injector (m3)'
1087:     ivarflag(223) = 2
1088:C
1089:     varname(224) = 'Vinj_d(t)'
1090:     vardesc(224) = 'volume of deuterium in pellet injector (m3)'
1091:     ivarflag(224) = 2
1092:C
1093:     varname(225) = 'Vinj_p(t)'
1094:     vardesc(225) = 'volume of protium in pellet injector (m3)'
1095:     ivarflag(225) = 2
1096:C
1097:     varname(226) = 'Xextru_t(t)'
1098:     vardesc(226) = 'mole fraction of tritium in pellet injector '
1099:    &          //'extruder'
1100:     ivarflag(226) = 2
1101:C
1102:     varname(227) = 'Xextru_d(t)'
1103:     vardesc(227) = 'mole fraction of deuterium in pellet injector '
1104:    &          //'extruder'
1105:     ivarflag(227) = 2
1106:C
1107:     varname(228) = 'Xextru_p(t)'
1108:     vardesc(228) = 'mole fraction of protium in pellet injector '
1109:    &          //'extruder'
1110:     ivarflag(228) = 2
1111:C
1112:     varname(229) = 'Nextru_(t)'
1113:     vardesc(229) = 'total amount in pellet injector extruder'
1114:     ivarflag(229) = 2
1115:C
1116:     varname(230) = 'Vinj_tot(t)'
1117:     vardesc(230) = 'total volume in pellet injector extruder'
1118:     ivarflag(230) = 2
1119:C
1120:     varname(231) = 'Dpellet(t)'
1121:     vardesc(231) = 'diameter of pellet injector pellet (m)'
1122:     ivarflag(231) = -1
1123:C
1124:     varname(232) = 'Hpellet(t)'
1125:     vardesc(232) = 'height of pellet injector pellet (m)'
1126:     ivarflag(232) = -1
1127:C
1128:     varname(233) = 'npellet_t(t)'
1129:     vardesc(233) = 'number of moles of tritium in pellet injector '
1130:    &          //'pellet (moles)'
1131:     ivarflag(233) = 3
1132:C
1133:     varname(234) = 'npellet_d(t)'
1134:     vardesc(234) = 'number of moles of deuterium in pellet injector '
1135:    &          //'pellet (moles)'
1136:     ivarflag(234) = 4
1137:C
```

page 14

```
1138:     varname(235) = 'npellet_p(t)'
1139:     vardesc(235) = 'number of moles of protium in pellet injector '
1140:    &          //'pellet (moles)'
1141:     ivarflag(235) = 2
1142:C
1143:     varname(236) = 'Rinj_(t)'
1144:     vardesc(236) = 'pellet repetition rate for pellet injector (Hz)'
1145:     ivarflag(236) = 4
1146:C
1147:     varname(237) = 'Finj_in_tot(t)'
1148:     vardesc(237) = 'total flow rate to plasma by pellet injector '
1149:    &          //'pellet (mole/h)'
1150:     ivarflag(237) = 4
1151:C
1152:     varname(238) = 'Fbk_plas_(t)'
1153:     vardesc(238) = 'backflow from plasma into pellet injector '
1154:    &          //'(mole/h)'
1155:     ivarflag(238) = 3
1156:C
1157:     varname(239) = 'Fbk_plas_t(t)'
1158:     vardesc(239) = 'tritium backflow from plasma into pellet '
1159:    &          //'injector (mole/h)'
1160:     ivarflag(239) = 3
1161:C
1162:     varname(240) = 'Fbk_plas_d(t)'
1163:     vardesc(240) = 'deuterium backflow from plasma into pellet '
1164:    &          //'injector (mole/h)'
1165:     ivarflag(240) = 3
1166:C
1167:     varname(241) = 'Fbk_plas_p(t)'
1168:     vardesc(241) = 'protium backflow from plasma into pellet '
1169:    &          //'injector (mole/h)'
1170:     ivarflag(241) = 4
1171:C
1172:     varname(242) = 'Fplas_prop(t)'
1173:     vardesc(242) = 'total flow rate to plasma from pellet injector '
1174:    &          //'propellant (mole/h)'
1175:     ivarflag(242) = 2
1176:C
1177:     varname(243) = 'Fplas_prop_t(t)'
1178:     vardesc(243) = 'tritium flow rate to plasma from pellet '
1179:    &          //'injector propellant (mole/h)'
1180:     ivarflag(243) = 2
1181:C
1182:     varname(244) = 'Fplas_prop_d(t)'
1183:     vardesc(244) = 'deuterium flow rate to plasma from pellet '
1184:    &          //'injector propellant (mole/h)'
1185:     ivarflag(244) = 3
1186:C
1187:     varname(245) = 'Fplas_prop_p(t)'
1188:     vardesc(245) = 'protium flow rate to plasma from pellet '
1189:    &          //'injector propellant (mole/h)'
1190:     ivarflag(245) = 3
1191:C
1192:     varname(246) = 'Finj_exh(t)'
1193:     vardesc(246) = 'exhaust flow rate from pellet injector (mole/h)'
1194:     ivarflag(246) = 3
1195:C
1196:     varname(247) = 'Finj_exh_t(t)'
1197:     vardesc(247) = 'tritium exhaust flow rate from pellet injector '
1198:    &          //'(mole/h)'
1199:     ivarflag(247) = 3
1200:C
1201:     varname(248) = 'Finj_exh_d(t)'
1202:     vardesc(248) = 'deuterium exhaust flow rate from pellet '
1203:    &          //'injector (mole/h)'
1204:     ivarflag(248) = 3
1205:C
1206:     varname(249) = 'Xinj_p(t)'
1207:     vardesc(249) = 'mole fraction of protium in pellet injector'
1208:     ivarflag(249) = 4
1209:C
1210:     varname(250) = 'Finj_prop(t)'
1211:     vardesc(250) = 'flow of propellant gas (mole/h)'
1212:     ivarflag(250) = 2
1213:C
1214:     varname(251) = 'Xprop_t(t)'
1215:     vardesc(251) = 'mole fraction of tritium in propellant gas flow '
1216:    &          //'(mole/h)'
1217:     ivarflag(251) = 2
1218:C
1219:     varname(252) = 'Xprop_d(t)'
1220:     vardesc(252) = 'mole fraction of deuterium in propellant gas '
1221:    &          //'flow (mole/h)'
1222:     ivarflag(252) = 3
1223:C
1224:     varname(253) = 'Xprop_p(t)'
1225:     vardesc(253) = 'mole fraction of protium in propellant gas flow '
1226:    &          //'(mole/h)'
1227:     ivarflag(253) = 3
1228:C
1229:     varname(254) = 'Finj_rec(t)'
1230:     vardesc(254) = 'recycle flow rate to sotope separation system '
1231:    &          //'from pellet injector (mole/h)'
1232:     ivarflag(254) = 4
1233:C
1234:     varname(255) = 'Fsto_tri_in(t)'
1235:     vardesc(255) = 'flow into tritium rich storage (mole/h)'
1236:     ivarflag(255) = 2
1237:C
1238:     varname(256) = 'Fsto_tri_out(t)'
1239:     vardesc(256) = 'flow out of tritium rich storage (mole/h)'
1240:     ivarflag(256) = 2
1241:C
1242:     varname(257) = 'Nsto_tri_(t)'
1243:     vardesc(257) = 'total amount in internal tritium rich storage '
1244:    &          //'(moles)'
1245:     ivarflag(257) = 1
1246:C
1247:     varname(258) = 'Tsto_tri_(t)'
1248:     vardesc(258) = 'total amount of tritium in internal tritium '
1249:    &          //'rich storage (moles)'
1250:     ivarflag(258) = 1
1251:C
```

```
1252:     varname(259) = 'Tsto_tri_m(t)'
1253:     vardesc(259) = '*mobilizable tritium in internal deuterium rich '
1254:    &         //'storage (moles)'
1255:     ivarflag(259) = 4
1256:C
1257:     varname(260) = 'Dsto_tri_(t)'
1258:     vardesc(260) = 'total amount of deuterium in internal tritium '
1259:    &         //'rich storage (moles)'
1260:     ivarflag(260) = 1
1261:C
1262:     varname(261) = 'Psto_tri_(t)'
1263:     vardesc(261) = 'total amount of protium in internal tritium '
1264:    &         //'rich storage (moles)'
1265:     ivarflag(261) = 4
1266:C
1267:     varname(262) = 'Xsto_tri_t(t)'
1268:     vardesc(262) = 'mole fraction of tritium in tritium rich storage'
1269:     ivarflag(262) = 2
1270:C
1271:     varname(263) = 'Xsto_tri_d(t)'
1272:     vardesc(263) = 'mole fraction of deuterium in tritium rich '
1273:    &         //'storage'
1274:     ivarflag(263) = 2
1275:C
1276:     varname(264) = 'Xsto_tri_p(t)'
1277:     vardesc(264) = 'mole fraction of protium in tritium rich storage'
1278:     ivarflag(264) = 3
1279:C
1280:     varname(265) = 'Fsto_deu_in(t)'
1281:     vardesc(265) = 'flow into deuterium rich storage (mole/h)'
1282:     ivarflag(265) = 2
1283:C
1284:     varname(266) = 'Fsto_deu_out(t)'
1285:     vardesc(266) = 'flow out of deuterium rich storage (mole/h)'
1286:     ivarflag(266) = 2
1287:C
1288:     varname(267) = 'Nsto_deu_(t)'
1289:     vardesc(267) = 'total amount in internal deuterium rich storage '
1290:    &         //'(moles)'
1291:     ivarflag(267) = 1
1292:C
1293:     varname(268) = 'Tsto_deu_(t)'
1294:     vardesc(268) = 'total amount of tritium in internal deuterium '
1295:    &         //'rich storage (moles)'
1296:     ivarflag(268) = 1
1297:C
1298:     varname(269) = 'Tsto_deu_m(t)'
1299:     vardesc(269) = 'mobilizable tritium in internal deuterium rich '
1300:    &         //'storage (moles)'
1301:     ivarflag(269) = 4
1302:C
1303:     varname(270) = 'Dsto_deu_(t)'
1304:     vardesc(270) = 'total amount of deuterium in internal deuterium '
1305:    &         //'rich storage (moles)'
1306:     ivarflag(270) = 1
1307:C
1308:     varname(271) = 'Psto_deu_(t)'
1309:     vardesc(271) = 'total amount of protium in deuterium rich '
1310:    &         //'storage (moles)'
1311:     ivarflag(271) = 4
1312:C
1313:     varname(272) = 'Ksto_tri_(t)'
1314:     vardesc(272) = 'accumulated inventory change in storage (mole)'
1315:     ivarflag(272) = 3
1316:C
1317:     varname(273) = 'Ksto_deu_(t)'
1318:     vardesc(273) = 'accumulated inventory change in storage (mole)'
1319:     ivarflag(273) = 3
1320:C
1321:     varname(274) = 'Ksto_pro_(t)'
1322:     vardesc(274) = 'accumulated inventory change in storage (mole)'
1323:     ivarflag(274) = 3
1324:C
1325:     varname(275) = 'Xsto_deu_t(t)'
1326:     vardesc(275) = 'mole fraction of tritium in deuterium rich '
1327:    &         //'storage'
1328:     ivarflag(275) = 2
1329:C
1330:     varname(276) = 'Xsto_deu_d(t)'
1331:     vardesc(276) = 'mole fraction of deuterium in deuterium rich '
1332:    &         //'storage'
1333:     ivarflag(276) = 2
1334:C
1335:     varname(277) = 'Xsto_deu_p(t)'
1336:     vardesc(277) = 'mole fraction of protium in deuterium rich '
1337:    &         //'storage'
1338:     ivarflag(277) = 3
1339:C
1340:     varname(278) = 'Fsto_pro_in(t)'
1341:     vardesc(278) = 'flow into protium rich storage (moles)'
1342:     ivarflag(278) = 2
1343:C
1344:     varname(279) = 'Fsto_pro_out(t)'
1345:     vardesc(279) = 'flow out of protium rich storage (moles)'
1346:     ivarflag(279) = 2
1347:C
1348:     varname(280) = 'Finj_sto_pro(t)'
1349:     vardesc(280) = 'flow from protium rich storage to pellet '
1350:    &         //'injector (mole/h)'
1351:     ivarflag(280) = -1
1352:C
1353:     varname(281) = 'Nsto_pro_(t)'
1354:     vardesc(281) = 'total amount in internal protium rich storage '
1355:    &         //'(moles)'
1356:     ivarflag(281) = 1
1357:C
1358:     varname(282) = 'Tsto_pro_(t)'
1359:     vardesc(282) = 'total amount of tritium in internal protium '
1360:    &         //'rich storage (moles)'
1361:     ivarflag(282) = 1
1362:C
1363:     varname(283) = 'Tsto_pro_m(t)'
1364:     vardesc(283) = 'mobilizable tritium in internal protium rich '
1365:    &         //'storage (moles)'
```

```
1366:     ivarflag(283) = 4
1367:C
1368:     varname(284) = 'Dsto_pro_(t)'
1369:     vardesc(284) = 'total amount of deuterium in internal protium '
1370:    &          //'rich storage (moles)'
1371:     ivarflag(284) = 1
1372:C
1373:     varname(285) = 'Psto_pro_(t)'
1374:     vardesc(285) = 'total amount of protium in internal protium '
1375:    &          //'rich storage (moles)'
1376:     ivarflag(285) = 3
1377:C
1378:     varname(286) = 'Xsto_pro_t(t)'
1379:     vardesc(286) = 'mole fraction of tritium in protium rich storage'
1380:     ivarflag(286) = 3
1381:C
1382:     varname(287) = 'Xsto_pro_d(t)'
1383:     vardesc(287) = 'mole fraction of deuterium in protium rich '
1384:    &          //'storage'
1385:     ivarflag(287) = 3
1386:C
1387:     varname(288) = 'Xsto_pro_p(t)'
1388:     vardesc(288) = 'mole fraction of protium in protium rich storage'
1389:     ivarflag(288) = 3
1390:C
1391:     varname(289) = 'Fsto_tri_out_t(t)'
1392:     vardesc(289) = 'required flow of tritium out of tritium rich '
1393:    &          //'storage (mole/h)'
1394:     ivarflag(289) = 4
1395:C
1396:     varname(290) = 'Fsto_deu_out_d(t)'
1397:     vardesc(290) = 'required flow of deuterium out of deuterium '
1398:    &          //'rich storage (mole/h)'
1399:     ivarflag(290) = 4
1400:C
1401:     varname(291) = 'Fsto_pro_out_p(t)'
1402:     vardesc(291) = 'required flow of protium outof protium rich '
1403:    &          //'flow (mole/h)'
1404:     ivarflag(291) = 4
1405:C
1406:C --- constants & descriptions
1407:C
1408:C
1409:     constname(1) = 'ap'
1410:     ap = 1.0D-2
1411:     constdesc(1) = 'relative combined reaction rates of proton '
1412:    &         // 'branch of DD reaction and D3H reaction'
1413:    &         // 'compared to DT'
1414:C
1415:     constname(2) = 'fbu'
1416:     fbu = 5.0D-2
1417:     constdesc(2) = 'fractional burn-up of tritium'
1418:C
1419:     constname(3) = 'fin_pellet'
1420:     fin_pellet = 5.0D-1
1421:     constdesc(3) = 'fraction of tritium fueling through pellet '
1422:    &          //'injection'
1423:C
1424:     constname(4) = 'fin_gas'
1425:     fin_gas = 7.0D-1
1426:     constdesc(4) = 'fraction of tritium fieling through gas puffing'
1427:C
1428:     constname(5) = 'Temperature'
1429:     Temperature = 1000
1430:     constdesc(5) = 'Diverter temperature (C degree)'
1431:C
1432:     constname(6) = 'Tedge'
1433:     Tedge = 60
1434:     constdesc(6) = 'Plasma Facing Material temperature (eV)'
1435:C
1436:     constname(7) = 'Medge'
1437:     Medge = 1
1438:     constdesc(7) = 'Plasma Facing Material (1/2/3=C/Be/W)'
1439:C
1440:     constname(8) = 'mu_A'
1441:     mu_A = 1.0D-1
1442:     constdesc(8) = 'mobility fraction of absorbed inventory'
1443:C
1444:     constname(9) = 'mu_C'
1445:     mu_C = 1.0D0
1446:     constdesc(9) = 'mobility fraction of codeposited inventory'
1447:C
1448:     constname(10) = 'mu_D'
1449:     mu_D = 1.0D0
1450:     constdesc(10) = 'mobility fraction of dust inventory'
1451:C
1452:     constname(11) = 'dc'
1453:     dc = 1.0D-1
1454:     constdesc(11) = 'dust concentration ratio (kg_tritium/kg_dust)'
1455:C
1456:     constname(12) = 'Vdiv'
1457:     Vdiv = (2.0D-5)*(2.0D5)
1458:     constdesc(12) = 'total volume of divertor (m3)'
1459:C
1460:     constname(13) = 'Vfw'
1461:     Vfw = (1.0D-3)*(1.6D3)*(2.0D-4)*(7.0D4)
1462:     constdesc(13) = 'total volume of first wall (m3)'
1463:C
1464:     constname(14) = 'Ad'
1465:     Ad = (1.0D-3)*(2.0D5)
1466:     constdesc(14) = 'divertor wall surface area (m2)'
1467:C
1468:     constname(15) = 'lambda_T'
1469:     lambda_T = 0.693/(12.4*365*24)
1470:     constdesc(15) = 'Decay constant of Tritium (1/h)'
1471:C
1472:     constname(16) = 'npump'
1473:     npump = 16
1474:     constdesc(16) = 'number of operating pumps'
1475:C
1476:     constname(18) = 'tpump'
1477:     tpump = 2.0D0*(2.0D0/3.0D0)
1478:     constdesc(18) = 'pumping for batch cryopumps in vacuum system (h)'
1479:C
```

page 17

```
1480:     constname(17) = 'treg'
1481:     treg = 2.0D0*(1.0D0/3.0D0)
1482:     constdesc(17) = 'regeneration time for batch cryopumps in vacuum '
1483:    &          //'system (h)'
1484:C
1485:     constname(19) = 'emr'
1486:     emr = 2.0D-1
1487:     constdesc(19) = 'efficiency of removal of cryodeposit for '
1488:    &          //'mechanical continuous regeneration cryopump'
1489:C
1490:     constname(20) = 'Rs'
1491:     Rs = 1.92D0
1492:     constdesc(20) = 'scraping rate for mechanical continuous '
1493:    &          //'regeneration scheme (m2/h)'
1494:C
1495:     constname(21) = 'Area_p'
1496:     Area_p = 3.2D-2
1497:     constdesc(21) = 'pump surface area for mechanical continuous '
1498:    &          //'regeneration scheme (m2).'
1499:C
1500:     constname(22) = 'mu_v'
1501:     mu_v = 1.0D-1
1502:     constdesc(22) = 'mobility fraction of inventory on cryopump.'
1503:C
1504:     constname(23) = 'mu_mob'
1505:     mu_mob = 1.0D-2
1506:     constdesc(23) = 'mobility fraction of inventory on fuel clean-up '
1507:    &          //'unit molecular sieve bed'
1508:C
1509:     constname(24) = 'Lhydro_'
1510:     Lhydro_ = 1.2D2
1511:     constdesc(24) = 'loading of mole sieves for hydrogen (scc/g)'
1512:C
1513:     constname(25) = 'Limp_'
1514:     Limp_ = 1.0D2
1515:     constdesc(25) = 'loading of mole sieves for impurities (scc/g)'
1516:C
1517:     constname(26) = 'Ns_max'
1518:     Ns_max = 1.77D1
1519:     constdesc(26) = 'maximum number of unfilled adsorption sites for '
1520:    &        // 'hydrogen on fuel clean-up unit'
1521:    &        // 'standby bed preloading with hydrogen (mole)'
1522:C
1523:     constname(27) = 'treg_msieve'
1524:     treg_msieve = 4.0D0
1525:     constdesc(27) = 'regeneration time for molecular sieve (h)'
1526:C
1527:     constname(28) = 'mu_reg_m'
1528:     mu_reg_m = 1.0D0
1529:     constdesc(28) = 'mobility fraction of inventory on fuel clean-up '
1530:    &          //'unit molecular sieve beds during regeneration'
1531:C
1532:     constname(29) = 'tau_pmem'
1533:     tau_pmem = 2.0D0
1534:     constdesc(29) = 'time constant for palladium membrane reactor (h)'
1535:C
1536:     constname(30) = 'w1'
1537:     w1 = 4.0D0
1538:     constdesc(30) = 'number of hydrogen atoms in a methane molecule'
1539:C
1540:     constname(31) = 'w2'
1541:     w2 = 3.0D0
1542:     constdesc(31) = 'number of hydrogen atoms in an ammonia molecule'
1543:C
1544:     constname(32) = 'w3'
1545:     w3 = 2.0D0
1546:     constdesc(32) = 'number of hydrogen atoms in a water molecule'
1547:C
1548:     constname(33) = 'xcq4_'
1549:     xcq4_ = 5.6D-1
1550:     constdesc(33) = 'specified mole fraction of methane in adjusted '
1551:    &          //'flow to purification'
1552:C
1553:     constname(34) = 'xnq3_'
1554:     xnq3_ = 4.0D-2
1555:     constdesc(34) = 'specified mole fraction of ammonia in adjusted '
1556:    &          //'flow to purification'
1557:C
1558:     constname(35) = 'xq2o_'
1559:     xq2o_ = 8.0D-2
1560:     constdesc(35) = 'specified mole fraction of water in adjusted '
1561:    &          //'flow to purification'
1562:C
1563:     constname(36) = 'Fbl_cool'
1564:     Fbl_cool = 1.0D-2
1565:     constdesc(36) = 'flow rate from coolant system (mole/h)'
1566:C
1567:     constname(37) = 'Fbl_'
1568:     Fbl_ = 5.0D-2
1569:     constdesc(37) = 'flow rate from blanket system (mole/h)'
1570:C
1571:     constname(38) = 'h1'
1572:     h1 = 3.0D-1
1573:     constdesc(38) = 'fraction of stream from fuel clean-up system '
1574:    &          //'not processed and directed to gas puffer'
1575:C
1576:     constname(39) = 'h3'
1577:     h3 = 3.0D-1
1578:     constdesc(39) = 'fraction of stream from fuel clean-up system '
1579:    &          //'not processed and directed to pellet injector'
1580:C
1581:     constname(40) = 'Fragnb'
1582:     Fragnb = 0.0D0
1583:     constdesc(40) = 'switch to include neutral beam in isotope '
1584:    &        // 'separation system balance (included if'
1585:    &        // 'equals one, not included if equals zero)'
1586:C
1587:     constname(41) = 'Fragiss_inj'
1588:     Fragiss_inj = 0.0D0
1589:     constdesc(41) = 'switch to include pellet injector in isotope '
1590:    &        // 'separation system balance (included if'
1591:    &        // 'equals one, not included if equals zero)'
1592:C
1593:     constname(42) = 'Xc_t'
```

page 18

```
1594:     Xc_t = 3.0D-1
1595:     constdesc(42) = 'mole fraction of tritium in coolant system'
1596:C
1597:     constname(43) = 'Xbl_t'
1598:     Xbl_t = 3.0D-1
1599:     constdesc(43) = 'mole fraction of tritium in breeder system'
1600:C
1601:     constname(44) = 'Xc_d'
1602:     Xc_d = 3.0D-1
1603:     constdesc(44) = 'mole fraction of deuterium in coolant stream'
1604:C
1605:     constname(45) = 'Xbl_d'
1606:     Xbl_d = 3.0D-1
1607:     constdesc(45) = 'mole fraction of deuterium in breeder stream'
1608:C
1609:     constname(46) = 'tau_iss_'
1610:     tau_iss_ = 2.22D0
1611:     constdesc(46) = 'residence time in isotope separation system (h)'
1612:C
1613:     constname(47) = 'mu_iss_'
1614:     mu_iss_ = 1.0D0
1615:     constdesc(47) = 'mobility fraction of inventory in isotope '
1616:    &          //'separation system'
1617:C
1618:     constname(48) = 'Xiss_top_t'
1619:     Xiss_top_t = 6.0D0/9.0D0
1620:     constdesc(48) = 'mole fraction of tritium from top of isotope '
1621:    &          //'separation system'
1622:C
1623:     constname(49) = 'Xiss_top_d'
1624:     Xiss_top_d = 2.0D0/9.0D0
1625:     constdesc(49) = 'mole fraction of deuterium from top of isotope '
1626:    &          //'separation system'
1627:C
1628:     constname(50) = 'Xiss_top_p'
1629:     Xiss_top_p = 1.0D0/9.0D0
1630:     constdesc(50) = 'mole fraction of tritium from top of isotope '
1631:    &          //'separation system'
1632:C
1633:     constname(51) = 'Xiss_mid_t'
1634:     Xiss_mid_t = 2.0D0/9.0D0
1635:     constdesc(51) = 'mole fraction of tritium from middle of isotope '
1636:    &          //'separation system'
1637:C
1638:     constname(52) = 'Xiss_mid_d'
1639:     Xiss_mid_d = 6.0D0/9.0D0
1640:     constdesc(52) = 'mole fraction of deuterium from middle of '
1641:    &          //'isotope separation system'
1642:C
1643:     constname(53) = 'Xiss_mid_p'
1644:     Xiss_mid_p = 1.0D0/9.0D0
1645:     constdesc(53) = 'mole fraction of tritium from middle of isotope '
1646:    &          //'separation system'
1647:C
1648:     constname(54) = 'Xiss_bot_t'
1649:     Xiss_bot_t = 1.0D0/9.0D0
1650:     constdesc(54) = 'mole fraction of tritium from bottom of isotope '
1651:    &          //'separation system'
1652:C
1653:     constname(55) = 'Xiss_bot_d'
1654:     Xiss_bot_d = 2.0D0/9.0D0
1655:     constdesc(55) = 'mole fraction of deuterium from bottom of '
1656:    &          //'isotope separation system'
1657:C
1658:     constname(56) = 'Xiss_bot_p'
1659:     Xiss_bot_p = 6.0D0/9.0D0
1660:     constdesc(56) = 'mole fraction of tritium from bottom of isotope '
1661:    &          //'separation system'
1662:C
1663:     constname(57) = 'eta_nb_tot'
1664:     eta_nb_tot = 3.1D-1
1665:     constdesc(57) = 'overall efficiency of transfer of neutral beam '
1666:    &          //'feed to the plasma'
1667:C
1668:     constname(58) = 'kiss_mid_'
1669:     kiss_mid_ = 3.0D-1
1670:     constdesc(58) = 'faction of flow from middle of isotope '
1671:    &          //'separation system directed to the neutral beam'
1672:C
1673:     constname(59) = 'fcryo_'
1674:     fcryo_ = 3.7D-1
1675:     constdesc(59) = 'fraction of feed to neutral beam that flows to '
1676:    &          //'cryopomp'
1677:C
1678:     constname(60) = 'tnb_reg'
1679:     tnb_reg = 2.0D0/3.0D0
1680:     constdesc(60) = 'regeneration time for batch neutral beam '
1681:    &          //'cryopumps (h)'
1682:C
1683:     constname(61) = 'tnb_pump'
1684:     tnb_pump = 2.0D0
1685:     constdesc(61) = 'pumping time for neutral beam batch '
1686:    &          //'regeneration cyopump (h)'
1687:C
1688:     constname(62) = 'nnb_'
1689:     nnb_ = 5.0D0
1690:     constdesc(62) = 'pumping cycle'
1691:C
1692:     constname(63) = 'mu_nb_mob'
1693:     mu_nb_mob = 1.0D0
1694:     constdesc(63) = 'mobility fraction of inventory on neutral beam '
1695:    &          //'cryopump'
1696:C
1697:     constname(64) = 'fnb_wall'
1698:     fnb_wall = 5.0D-2
1699:     constdesc(64) = 'fraction of feed to neutral beam that ends up '
1700:    &          //'on the walls'
1701:C
1702:     constname(65) = 'mu_wall_'
1703:     mu_wall_ = 1.0D0
1704:     constdesc(65) = 'mobility fraction of inventory in neutral beam '
1705:    &          //'walls'
1706:C
1707:     constname(66) = 'tcycle_p'
```

page 19

```
1708:     tcycle_p = 2200d0/3600d0
1709:     constdesc(66) = 'plasma burning time (h)'
1710:C
1711:     constname(67) = 't_p_ru'
1712:     t_p_ru = 50d0/3600d0
1713:     constdesc(67) = 'ramp-up time for plasma (h)'
1714:C
1715:     constname(68) = 't_p_rd'
1716:     t_p_rd = 100d0/3600d0
1717:     constdesc(68) = 'ramp-down time for plasma (h)'
1718:C
1719:     constname(69) = 't_nb_ru'
1720:     t_nb_ru = 100d0/3600d0
1721:     constdesc(69) = 'ramp-up time for neutral beam (h)'
1722:C
1723:     constname(70) = 't_nb_rd'
1724:     t_nb_rd = 100d0/3600d0
1725:     constdesc(70) = 'ramp-down time for neutral beam (h)'
1726:C
1727:     constname(71) = 't_burn'
1728:     t_burn = 1000d0/3600d0
1729:     constdesc(71) = 'plasma burn time (h)'
1730:C
1731:     constname(72) = 'tcycle_nb'
1732:c    tcycle_nb = (100d0+600d0+100d0)/3600d0
1733:     tcycle_nb = (50d0+1000d0+100d0)/3600d0
1734:     constdesc(72) = 'cycle time for neutral beam (h)'
1735:C
1736:     constname(73) = 'E_nb'
1737:     E_nb = 1.3D0
1738:     constdesc(73) = 'neutral beam particle energy (MeV)'
1739:C
1740:     constname(74) = 'No'
1741:     No = 6.022D23
1742:     constdesc(74) = 'Avogadros number'
1743:C
1744:     constname(75) = 'nedge_tot'
1745:     nedge_tot = (1.0D20 + 1.0D20 + 1.0D20*5D-4)
1746:     constdesc(75) = ''
1747:C
1748:     constname(76) = 'nedge_t'
1749:     nedge_t = 1.0D20
1750:     constdesc(76) = ''
1751:C
1752:     constname(77) = 'nedge_d'
1753:     nedge_d = 1.0D20
1754:     constdesc(77) = ''
1755:C
1756:     constname(78) = 'nedge_p'
1757:     nedge_p = 1.0D20 * 5.0D-4
1758:     constdesc(78) = ''
1759:C
1760:     constname(79) = 'mgas_mid_iss'
1761:     mgas_mid_iss = 3.0D-1
1762:     constdesc(79) = 'fraction of flow from middle of isotope '
1763:    &          //'separation system directed to gas puffer'
1764:C
1765:     constname(80) = 'mgas_bot_iss'
1766:     mgas_bot_iss = 3.0D-1
1767:     constdesc(80) = 'fraction of flow from bottom of isotope '
1768:    &          //'separation system directed to gas puffer'
1769:C
1770:     constname(81) = 'mu_gas'
1771:     mu_gas = 1.0D0
1772:     constdesc(81) = 'mobility fraction for gas puffer'
1773:C
1774:     constname(82) = 'Vpellet'
1775:     Vpellet = 4.5D-7
1776:     constdesc(82) = 'volume of pellet injector pellet (m3)'
1777:C
1778:     constname(83) = 'finj_err'
1779:     finj_err = 3.0D-1
1780:     constdesc(83) = 'erosion fraction of deuterium and tritium for '
1781:    &          //'pellet injector'
1782:C
1783:     constname(84) = 'linj_mid_iss'
1784:     linj_mid_iss = 3.0D-1
1785:     constdesc(84) = 'fraction of flow from middle of isotope '
1786:    &          //'separation system to pellet injector'
1787:C
1788:     constname(85) = 'linj_bot_iss'
1789:     linj_bot_iss = 3.0D-1
1790:     constdesc(85) = 'fraction of flow from bottom of isotope '
1791:    &          //'separation system to pellet injector'
1792:C
1793:     constname(86) = 'mu_ext'
1794:     mu_ext = 1.0D0
1795:     constdesc(86) = 'mobility fraction of inventory in extruders'
1796:C
1797:     constname(87) = 'kpellet_t'
1798:     kpellet_t = 2*5.294D4
1799:     constdesc(87) = 'mole composition of tritium per unit volume of '
1800:    &          //'pellet injector pellet'
1801:C
1802:     constname(88) = 'kpellet_d'
1803:     kpellet_d = 2*4.979D4
1804:     constdesc(88) = 'mole composition of deuterium per unit volume '
1805:    &          //'of pellet injector pellet'
1806:C
1807:     constname(89) = 'kpellet_p'
1808:     kpellet_p = 2*4.287D4
1809:     constdesc(89) = 'mole composition of protium per unit volume of '
1810:    &          //'pellet injector pellet'
1811:C
1812:     constname(90) = 'fcap_prop'
1813:     fcap_prop = 9.996D-1
1814:     constdesc(90) = 'farction of propellant recaptured before entry '
1815:    &          //'into plasma'
1816:C
1817:     constname(91) = 'fnon_iss'
1818:     fnon_iss = 0.7D0
1819:     constdesc(91) = 'fraction of moles in pellet injector exhaust '
1820:    &        // 'stream not processd bu the isotope'
1821:    &        // 'separation system'
```

page 20

```
1822:C
1823:     constname(92) = 'liss_p'
1824:     liss_p = 3.2D-2
1825:     constdesc(92) = 'fraction of flow from top of isotope separation '
1826:    &          //'system directed to pellet injector'
1827:C
1828:     constname(93) = 'nprop'
1829:     nprop = 4.45D-1
1830:     constdesc(93) = 'amount of propellant gas required per pellet '
1831:    &          //'for pellet injector (moles)'
1832:C
1833:     constname(94) = 'mu_sto'
1834:     mu_sto = 1.0D-2
1835:     constdesc(94) = 'mobility fraction on storage inventories'
1836:C
1837:     constname(95) = 'linj_p'
1838:     linj_p = 3.2D-2
1839:     constdesc(95) = 'fraction of fliw from top of isotope separation '
1840:    &          //'system directed to pellet injector'
1841:C
1842:     constname(96) = 't_dwell'
1843:     t_dwell = 1050d0/3600d0
1844:     constdesc(96) = 'dwelling time for plasma (h)'
1845:C
1846:     constname(97) = 'volume'
1847:     volume = 1.6d9
1848:     constdesc(97) = 'reactor volume  (cm3)'
1849:C
1850:     constname(98) = 'Namsb_max'
1851:     Namsb_max = 9.1d0
1852:     constdesc(98) = '  (moles)'
1853:C
1854:     constname(99) = 'Ncmsb_max'
1855:     Ncmsb_max = 10.2d0
1856:     constdesc(99) = '  (moles)'
1857:C
1858:C --- idfvar(ndfvar) ...
1859:C
1860:
1861:C  ... C_t
1862:     idfvar(1) = 21
1863:C  ... C_d
1864:     idfvar(2) = 29
1865:C  ... C_p
1866:     idfvar(3) = 37
1867:C  ... D_t
1868:     idfvar(4) = 23
1869:C  ... D_d
1870:     idfvar(5) = 31
1871:C  ... D_p
1872:     idfvar(6) = 39
1873:C  ... Tpv
1874:     idfvar(7) = 45
1875:C  ... Dpv
1876:     idfvar(8) = 50
1877:C  ... Ppv
1878:     idfvar(9) = 57
1879:C  ... Hepv
1880:     idfvar(10) = 59
1881:C  ... Nimpt
1882:     idfvar(11) = 81
1883:C  ... Nnhydro_imp
1884:     idfvar(12) = 83
1885:C  ... Ncq4_imp
1886:     idfvar(13) = 84
1887:C  ... Nnq3_imp
1888:     idfvar(14) = 85
1889:C  ... Nq2o_imp
1890:     idfvar(15) = 86
1891:C  ... Timp_
1892:     idfvar(16) = 87
1893:C  ... Dimp_
1894:     idfvar(17) = 88
1895:C  ... Pimp_
1896:     idfvar(18) = 89
1897:C  ... Ns
1898:     idfvar(19) = 95
1899:C  ... T2_sub
1900:     idfvar(20) = 99
1901:C  ... D2_sub
1902:     idfvar(21) = 100
1903:C  ... P2_sub
1904:     idfvar(22) = 101
1905:C  ... Tpmem
1906:     idfvar(23) = 115
1907:C  ... Dpmem
1908:     idfvar(24) = 116
1909:C  ... Ppmem
1910:     idfvar(25) = 117
1911:C  ... Niss_
1912:     idfvar(26) = 143
1913:C  ... Tiss_
1914:     idfvar(27) = 144
1915:C  ... Diss_
1916:     idfvar(28) = 146
1917:C  ... Ncryo_
1918:     idfvar(29) = 159
1919:C  ... Tnb_cryo
1920:     idfvar(30) = 163
1921:C  ... Dnb_cryo
1922:     idfvar(31) = 170
1923:C  ... Tnb_wall
1924:     idfvar(32) = 178
1925:C  ... Next_12
1926:     idfvar(33) = 211
1927:C  ... Text_12
1928:     idfvar(34) = 212
1929:C  ... Dext_12
1930:     idfvar(35) = 213
1931:C  ... Ngas_tot
1932:     idfvar(36) = 195
1933:C  ... Tgas_
1934:     idfvar(37) = 197
1935:C  ... Dgas_
```

```
1936:     idfvar(38) = 198
1937:C  ... Nsto_tri_
1938:     idfvar(39) = 257
1939:C  ... Tsto_tri_
1940:     idfvar(40) = 258
1941:C  ... Dsto_tri_
1942:     idfvar(41) = 260
1943:C  ... Nsto_deu_
1944:     idfvar(42) = 267
1945:C  ... Tsto_deu_
1946:     idfvar(43) = 268
1947:C  ... Dsto_deu_
1948:     idfvar(44) = 270
1949:C  ... Nsto_pro_
1950:     idfvar(45) = 281
1951:C  ... Tsto_pro_
1952:     idfvar(46) = 282
1953:C  ... Dsto_pro_
1954:     idfvar(47) = 284
1955:     ierr = 0
1956:
1957:     if(lpr.ge.1) then
1958:       write(6,*) '*** variables ***'
1959:       do i=1,nvar
1960:         write(6,*) 'varname(',i,') ',varname(i)
1961:       end do
1962:     end if
1963:     if(lpr.eq.2) then
1964:       write(6,*) '*** constants ***'
1965:       do i=1,nconst
1966:         write(6,*) constname(i),': ',C(i)
1967:       end do
1968:     end if
1969:
1970:     return
1971:     end
```

● varinit.f

```
 1:     subroutine varinit(delta_t,F,pr,jerr)
 2:C
 3:C ***************************************************
 4:C   Automatically created.
 5:C   creation date : << Tue May 6 14:21:39 1997 >>
 6:C ***************************************************
 7:     implicit double precision (a-h,o-z)
 8:C
 9:C ... initial condition ...
10:C
11:
12:     include '_sizes.h'
13:     include 'varconst.h'
14:     integer pr,jerr
15:     double precision F(ndfvar),delta_t
16:     double precision fppower0
17:
18:C ... statement function ...
19:     ABX(AA,BB,XX) = AA+ BB*XX
20:
21:C ... jerr(if=0>noerror,if=1>error)
22:C ... pr(if=0>not print,if=1>print)
23:
24:     jerr = 0
25:
26:C
27:C  Initial_step = 0 is necessary for varinit.
28:C
29:     Initial_step = 0
30:     t = 0.0D0
31:     j = 0
32:C ... initials of X(n) ...
33:
34:*
35:* --- Storage ---------------------------------------------
36:*
37:
38:
39:c****    delt0 = 20.d0 / 3600d0
40:c****    delt0 = 10.d0 / 3600d0
41:     delt0 = t_p_ru / 10.d0
42:
43:     Burn_t(0) = 2.12D-3 * 1000 * delt0 / t_p_ru
44:     if(pr.ge.2) write(6,*) 'Burn_t=',Burn_t(0)
45:
46:     Burn_d(0) = Burn_t(0)
47:     if(pr.ge.2) write(6,*) 'Burn_d=',Burn_d(0)
48:
49:     Reac_p(0) = ( 0.01D0 * 0.05 ) * Burn_t(0)
50:     if(pr.ge.2) write(6,*) 'Reac_p=',Reac_p(0)
51:
52:     ttstep = 20d0 / 3600
53:
54:     Tsto_tri_(0) = Burn_t(0) * ttstep / fbu
55:     if(pr.ge.2) write(6,*) 'Tsto_tri_=',Tsto_tri_(0)
56:
57:     Dsto_tri_(0) = 1.0d-3 * Tsto_tri_(0)
58:     if(pr.ge.2) write(6,*) 'Dsto_tri_=',Dsto_tri_(0)
59:
60:     Psto_tri_(0) = 1.0d-3 * Tsto_tri_(0)
61:     if(pr.ge.2) write(6,*) 'Psto_tri_=',Psto_tri_(0)
62:
63:     Nsto_tri_(0) = Tsto_tri_(0) + Dsto_tri_(0) + Psto_tri_(0)
64:     if(pr.ge.2) write(6,*) 'Nsto_tri_=',Nsto_tri_(0)
65:
66:     Tsto_tri_m(0) = mu_sto * Tsto_tri_(0)
67:     if(pr.ge.2) write(6,*) 'Tsto_tri_m=',Tsto_tri_m(0)
68:
69:     fsto_tri_out(0) = Burn_t(0)
```

```
70:     if(pr.ge.2) write(6,*) 'Fsto_tri_out=',Fsto_tri_out(0)
71:*Fsto_tri_in  = (1.0D0 - linj_bot_iss - mgas_bot_iss) * Fiss_bot_ ;
72:
73:     Ksto_tri_(0) = 0.0d0
74:     if(pr.ge.2) write(6,*) 'Ksto_tri_=',Ksto_tri_(0)
75:
76:     Ksto_deu_(0) = 0.0d0
77:     if(pr.ge.2) write(6,*) 'Ksto_deu_=',Ksto_deu_(0)
78:
79:     Ksto_pro_(0) = 0.0d0
80:     if(pr.ge.2) write(6,*) 'Ksto_pro_=',Ksto_pro_(0)
81:
82:     Xsto_tri_t(0) = Tsto_tri_(0) / Nsto_tri_(0)
83:     if(pr.ge.2) write(6,*) 'Xsto_tri_t=',Xsto_tri_t(0)
84:
85:     Xsto_tri_d(0) = Dsto_tri_(0) / Nsto_tri_(0)
86:     if(pr.ge.2) write(6,*) 'Xsto_tri_d=',Xsto_tri_d(0)
87:
88:     Xsto_tri_p(0) = 1.0D0 - Xsto_tri_t(0) - Xsto_tri_d(0)
89:     if(pr.ge.2) write(6,*) 'Xsto_tri_p=',Xsto_tri_p(0)
90:
91:     Dsto_deu_(0) = Burn_d(0) * ttstep / fbu
92:     if(pr.ge.2) write(6,*) 'Dsto_deu_=',Dsto_deu_(0)
93:
94:     Tsto_deu_(0) = 1.0d-3 * Dsto_deu_(0)
95:     if(pr.ge.2) write(6,*) 'Tsto_deu_=',Tsto_deu_(0)
96:
97:     Psto_deu_(0) = 1.0d-3 * Dsto_deu_(0)
98:     if(pr.ge.2) write(6,*) 'Psto_deu_=',Psto_deu_(0)
99:
100:    Nsto_deu_(0) = Tsto_deu_(0) + Dsto_deu_(0) + Psto_deu_(0)
101:    if(pr.ge.2) write(6,*) 'Nsto_deu_=',Nsto_deu_(0)
102:
103:    Tsto_deu_m(0) = mu_sto * Tsto_deu_(0)
104:    if(pr.ge.2) write(6,*) 'Tsto_deu_m=',Tsto_deu_m(0)
105:
106:    Fsto_deu_out(0) = Burn_d(0)
107:    if(pr.ge.2) write(6,*) 'Fsto_deu_out=',Fsto_deu_out(0)
108:
109:    Fgas_in_deu(0) = Fsto_deu_out(0) / 4
110:    if(pr.ge.2) write(6,*) 'Fgas_in_deu=',Fgas_in_deu(0)
111:*Fsto_deu_in  = (1.0D0 -0.3D0 -0.3D0 -0.3D0) * Fiss_mid_ ;
112:
113:    Xsto_deu_t(0) = Tsto_deu_(0) / Nsto_deu_(0)
114:    if(pr.ge.2) write(6,*) 'Xsto_deu_t=',Xsto_deu_t(0)
115:
116:    Xsto_deu_d(0) = Dsto_deu_(0) / Nsto_deu_(0)
117:    if(pr.ge.2) write(6,*) 'Xsto_deu_d=',Xsto_deu_d(0)
118:
119:    Xsto_deu_p(0) = 1.0D0 - Xsto_deu_t(0) - Xsto_deu_d(0)
120:    if(pr.ge.2) write(6,*) 'Xsto_deu_p=',Xsto_deu_p(0)
121:
122:    Psto_pro_(0) = Reac_p(0) * ttstep / fbu
123:    if(pr.ge.2) write(6,*) 'Psto_pro_=',Psto_pro_(0)
124:
125:    Tsto_pro_(0) = 1.0d-3 * Psto_pro_(0)
126:    if(pr.ge.2) write(6,*) 'Tsto_pro_=',Tsto_pro_(0)
127:
128:    Dsto_pro_(0) = 1.0d-3 * Psto_pro_(0)
129:    if(pr.ge.2) write(6,*) 'Dsto_pro_=',Dsto_pro_(0)
130:
131:    Nsto_pro_(0) = Tsto_pro_(0) + Dsto_pro_(0) + Psto_pro_(0)
132:    if(pr.ge.2) write(6,*) 'Nsto_pro_=',Nsto_pro_(0)
133:
134:    Tsto_pro_m(0) = mu_sto * Tsto_pro_(0)
135:    if(pr.ge.2) write(6,*) 'Tsto_pro_m=',Tsto_pro_m(0)
136:*Fsto_pro_in  = ( 1.0D0 - 0.032D0 ) * Fiss_top_ ;
137:
138:    Fsto_pro_out(0) = Reac_p(0)
139:    if(pr.ge.2) write(6,*) 'Fsto_pro_out=',Fsto_pro_out(0)
140:
141:    Fsto_out_p(0) = Fsto_pro_out(0)
142:    if(pr.ge.2) write(6,*) 'Fsto_out_p=',Fsto_out_p(0)
143:
144:    Xsto_pro_t(0) = Tsto_pro_(0) / Nsto_pro_(0)
145:    if(pr.ge.2) write(6,*) 'Xsto_pro_t=',Xsto_pro_t(0)
146:
147:    Xsto_pro_d(0) = Dsto_pro_(0) / Nsto_pro_(0)
148:    if(pr.ge.2) write(6,*) 'Xsto_pro_d=',Xsto_pro_d(0)
149:
150:    Xsto_pro_p(0) = Psto_pro_(0) / Nsto_pro_(0)
151:    if(pr.ge.2) write(6,*) 'Xsto_pro_p=',Xsto_pro_p(0)
152:
153:    Fsto_tri_out_t(0) = Fsto_tri_out(0) * Xsto_tri_t(0)
154:    if(pr.ge.2) write(6,*) 'Fsto_tri_out_t=',Fsto_tri_out_t(0)
155:
156:    Fsto_deu_out_d(0) = Fsto_deu_out(0) * Xsto_tri_d(0)
157:    if(pr.ge.2) write(6,*) 'Fsto_deu_out_d=',Fsto_deu_out_d(0)
158:*
159:* --- Fueling ------------------------------------------------
160:*
161:
162:    Fuelf_t(0) = Burn_t(0) / fbu
163:    if(pr.ge.2) write(6,*) 'Fuelf_t=',Fuelf_t(0)
164:*
165:* --- The Plasma ---------------------------------------------
166:*
167:
168:    Pfusion(0) = Burn_t(0) / 2.12D-3
169:    if(pr.ge.2) write(6,*) 'Pfusion=',Pfusion(0)
170:
171:    Fuel_t(0) = Burn_t(0) / fbu
172:    if(pr.ge.2) write(6,*) 'Fuel_t=',Fuel_t(0)
173:
174:    Fex_t(0) = Fuel_t(0) * ( 1 - fbu )
175:    if(pr.ge.2) write(6,*) 'Fex_t=',Fex_t(0)
176:
177:    Fplas_in_t(0) = Fuelf_t(0) * fin_pellet
178:    if(pr.ge.2) write(6,*) 'Fplas_in_t=',Fplas_in_t(0)
179:
180:    Fgas_t(0) = Fuelf_t(0) * fin_gas
181:    if(pr.ge.2) write(6,*) 'Fgas_t=',Fgas_t(0)
182:
183:    Fplas_in_d(0) = Fuelf_t(0) * fin_pellet
```

page 23

```
184:     if(pr.ge.2) write(6,*) 'Fplas_in_d=',Fplas_in_d(0)
185:
186:     Fgas_t(0) = Fuelf_t(0) * fin_gas
187:     if(pr.ge.2) write(6,*) 'Fgas_t=',Fgas_t(0)
188:*** Fuel_d = Fplas_in_d + Fgas_d + Fplas_nb_d + Fplas_prop_d ;
189:*** Fex_d      = Fuel_d - Burn_d ;
190:
191:     Fhe_ex(0) = Burn_t(0)
192:     if(pr.ge.2) write(6,*) 'Fhe_ex=',Fhe_ex(0)
193:*
194:* --- Plasma Facing Components -------------------------------
195:*
196:
197:     i_t(0) = 0.4D0
198:     if(pr.ge.2) write(6,*) 'i_t=',i_t(0)
199:
200:     i_d(0) = 0.3D0
201:     if(pr.ge.2) write(6,*) 'i_d=',i_d(0)
202:
203:     i_p(0) = 0.3D0
204:     if(pr.ge.2) write(6,*) 'i_p=',i_p(0)
205:
206:     fmMVdiv = fm ( Medge ) * ai ( Temperature , 0d0 ) * Vdiv / ( 0.5
207:    &      * 3.016D0 )
208:
209:     fmMVfw = fm ( Medge ) * ai ( Temperature , 0d0 ) * Vfw / ( 0.5 *
210:    &      3.016D0 )
211:
212:     fppp = delt0 / t_p_ru
213:
214:     Adiv_t(0) = fmMVdiv * i_t(0) * fppp
215:     if(pr.ge.2) write(6,*) 'Adiv_t=',Adiv_t(0)
216:
217:     Afw_t(0) = fmMVfw * i_t(0) * fppp
218:     if(pr.ge.2) write(6,*) 'Afw_t=',Afw_t(0)
219:
220:     Adiv_d(0) = fmMVdiv * i_d(0) * fppp
221:     if(pr.ge.2) write(6,*) 'Adiv_d=',Adiv_d(0)
222:
223:     Afw_d(0) = fmMVfw * i_d(0) * fppp
224:     if(pr.ge.2) write(6,*) 'Afw_d=',Afw_d(0)
225:
226:     Adiv_p(0) = fmMVdiv * i_p(0) * fppp
227:     if(pr.ge.2) write(6,*) 'Adiv_p=',Adiv_p(0)
228:
229:     Afw_p(0) = fmMVfw * i_p(0) * fppp
230:     if(pr.ge.2) write(6,*) 'Afw_p=',Afw_p(0)
231:
232:     d_Adiv_t(0) = 0.0D0
233:     if(pr.ge.2) write(6,*) 'd_Adiv_t=',d_Adiv_t(0)
234:
235:     d_Adiv_d(0) = 0.0D0
236:     if(pr.ge.2) write(6,*) 'd_Adiv_d=',d_Adiv_d(0)
237:
238:     d_Adiv_p(0) = 0.0D0
239:     if(pr.ge.2) write(6,*) 'd_Adiv_p=',d_Adiv_p(0)
240:
241:     d_Afw_t(0) = 0.0D0
242:     if(pr.ge.2) write(6,*) 'd_Afw_t=',d_Afw_t(0)
243:
244:     d_Afw_d(0) = 0.0D0
245:     if(pr.ge.2) write(6,*) 'd_Afw_d=',d_Afw_d(0)
246:
247:     d_Afw_p(0) = 0.0D0
248:     if(pr.ge.2) write(6,*) 'd_Afw_p=',d_Afw_p(0)
249:
250:     C_t(0) = 0.0D0
251:     if(pr.ge.2) write(6,*) 'C_t=',C_t(0)
252:
253:     C_d(0) = 0.0D0
254:     if(pr.ge.2) write(6,*) 'C_d=',C_d(0)
255:
256:     C_p(0) = 0.0D0
257:     if(pr.ge.2) write(6,*) 'C_p=',C_p(0)
258:
259:     D_t(0) = 0.0D0
260:     if(pr.ge.2) write(6,*) 'D_t=',D_t(0)
261:
262:     D_d(0) = 0.0D0
263:     if(pr.ge.2) write(6,*) 'D_d=',D_d(0)
264:
265:     D_p(0) = 0.0D0
266:     if(pr.ge.2) write(6,*) 'D_p=',D_p(0)
267:
268:     d_C_t(0) = 0.0D0
269:     if(pr.ge.2) write(6,*) 'd_C_t=',d_C_t(0)
270:
271:     d_C_d(0) = 0.0D0
272:     if(pr.ge.2) write(6,*) 'd_C_d=',d_C_d(0)
273:
274:     d_C_p(0) = 0.0D0
275:     if(pr.ge.2) write(6,*) 'd_C_p=',d_C_p(0)
276:
277:     d_D_t(0) = 0.0D0
278:     if(pr.ge.2) write(6,*) 'd_D_t=',d_D_t(0)
279:
280:     d_D_d(0) = 0.0D0
281:     if(pr.ge.2) write(6,*) 'd_D_d=',d_D_d(0)
282:
283:     d_D_p(0) = 0.0D0
284:     if(pr.ge.2) write(6,*) 'd_D_p=',d_D_p(0)
285:
286:     Tpfc_(0) = Adiv_t(0) + Afw_t(0) + C_t(0) + D_t(0)
287:     if(pr.ge.2) write(6,*) 'Tpfc_=',Tpfc_(0)
288:
289:     Tpfc_m(0) = mu_A * ( Adiv_t(0) + Afw_t(0) ) + mu_C * C_t(0) +
290:    &      mu_D * D_t(0)
291:     if(pr.ge.2) write(6,*) 'Tpfc_m=',Tpfc_m(0)
292:
293:     Dpfc_(0) = Adiv_d(0) + Afw_d(0) + C_d(0) + D_d(0)
294:     if(pr.ge.2) write(6,*) 'Dpfc_=',Dpfc_(0)
295:
296:     Ppfc_(0) = Adiv_p(0) + Afw_p(0) + C_p(0) + D_p(0)
297:     if(pr.ge.2) write(6,*) 'Ppfc_=',Ppfc_(0)
```

```
298:*
299:* --- Vacuum -------------------------------
300:*
301:*
302:
303:      Tpv(0) = 1.0D-4
304:      if(pr.ge.2) write(6,*) 'Tpv=',Tpv(0)
305:
306:      Dpv(0) = 1.0D-4
307:      if(pr.ge.2) write(6,*) 'Dpv=',Dpv(0)
308:
309:      Ppv(0) = 1.0D-4
310:      if(pr.ge.2) write(6,*) 'Ppv=',Ppv(0)
311:
312:      Hepv(0) = 1.0D-4
313:      if(pr.ge.2) write(6,*) 'Hepv=',Hepv(0)
314:
315:      emrRsAp = emr * Rs / Area_p
316:
317:      Rt_rem(0) = Tpv(0) * emrRsAp
318:      if(pr.ge.2) write(6,*) 'Rt_rem=',Rt_rem(0)
319:
320:      Rd_rem(0) = Dpv(0) * emrRsAp
321:      if(pr.ge.2) write(6,*) 'Rd_rem=',Rd_rem(0)
322:
323:      Rp_rem(0) = Ppv(0) * emrRsAp
324:      if(pr.ge.2) write(6,*) 'Rp_rem=',Rp_rem(0)
325:
326:      Rhe_rem(0) = Hepv(0) * emrRsAp
327:      if(pr.ge.2) write(6,*) 'Rhe_rem=',Rhe_rem(0)
328:
329:      Tp_reg(0) = Rt_rem(0) * treg
330:      if(pr.ge.2) write(6,*) 'Tp_reg=',Tp_reg(0)
331:
332:      Dp_reg(0) = Rd_rem(0) * treg
333:      if(pr.ge.2) write(6,*) 'Dp_reg=',Dp_reg(0)
334:
335:      Pp_reg(0) = Rp_rem(0) * treg
336:      if(pr.ge.2) write(6,*) 'Pp_reg=',Pp_reg(0)
337:
338:      Hep_reg(0) = Rhe_rem(0) * treg
339:      if(pr.ge.2) write(6,*) 'Hep_reg=',Hep_reg(0)
340:
341:      Tp_tot(0) = Tpv(0) * npump
342:      if(pr.ge.2) write(6,*) 'Tp_tot=',Tp_tot(0)
343:
344:      Dp_tot(0) = Dpv(0) * npump
345:      if(pr.ge.2) write(6,*) 'Dp_tot=',Dp_tot(0)
346:
347:      Pp_tot(0) = Ppv(0) * npump
348:      if(pr.ge.2) write(6,*) 'Pp_tot=',Pp_tot(0)
349:
350:      Hep_tot(0) = Hepv(0) * npump
351:      if(pr.ge.2) write(6,*) 'Hep_tot=',Hep_tot(0)
352:
353:      Tp_tot_m(0) = mu_v * Tp_tot(0)
354:      if(pr.ge.2) write(6,*) 'Tp_tot_m=',Tp_tot_m(0)
355:
356:      Fv_out(0) = ( Rt_rem(0) + Rd_rem(0) + Rp_rem(0) + Rhe_rem(0) ) *
357:     &      npump
358:      if(pr.ge.2) write(6,*) 'Fv_out=',Fv_out(0)
359:
360:      Fpuri_in(0) = Fv_out(0)
361:      if(pr.ge.2) write(6,*) 'Fpuri_in=',Fpuri_in(0)
362:
363:      FF = Rt_rem(0) + Rd_rem(0) + Rp_rem(0)
364:
365:      if ( FF .gt. 0.0d0 ) then
366:
367:      it_(0) = Rt_rem(0) / FF
368:      if(pr.ge.2) write(6,*) 'it_=',it_(0)
369:
370:      id_(0) = Rd_rem(0) / FF
371:      if(pr.ge.2) write(6,*) 'id_=',id_(0)
372:
373:      ip_(0) = Rp_rem(0) / FF
374:      if(pr.ge.2) write(6,*) 'ip_=',ip_(0)
375:
376:      else
377:
378:      it_(0) = 0.0d0
379:      if(pr.ge.2) write(6,*) 'it_=',it_(0)
380:
381:      id_(0) = 0.0d0
382:      if(pr.ge.2) write(6,*) 'id_=',id_(0)
383:
384:      ip_(0) = 0.0d0
385:      if(pr.ge.2) write(6,*) 'ip_=',ip_(0)
386:
387:      endif
388:
389:      Fadj_puri(0) = Fv_out(0)
390:      if(pr.ge.2) write(6,*) 'Fadj_puri=',Fadj_puri(0)
391:
392:      Fpure_(0) = Fv_out(0) * 0.98
393:      if(pr.ge.2) write(6,*) 'Fpure_=',Fpure_(0)
394:
395:      Fimp_(0) = Fv_out(0) * 0.02
396:      if(pr.ge.2) write(6,*) 'Fimp_=',Fimp_(0)
397:
398:      Fcq4_imp(0) = xcq4_ * Fv_out(0) * 0.02
399:      if(pr.ge.2) write(6,*) 'Fcq4_imp=',Fcq4_imp(0)
400:
401:      Fnq3_imp(0) = xnq3_ * Fv_out(0) * 0.02
402:      if(pr.ge.2) write(6,*) 'Fnq3_imp=',Fnq3_imp(0)
403:
404:      Fq2o_imp(0) = xq2o_ * Fv_out(0) * 0.02
405:      if(pr.ge.2) write(6,*) 'Fq2o_imp=',Fq2o_imp(0)
406:
407:      xh_imp(0) = xcq4_ + xnq3_ + xq2o_
408:      if(pr.ge.2) write(6,*) 'xh_imp=',xh_imp(0)
409:
410:      xnhydro_(0) = 1 - 0.0136d0
411:      if(pr.ge.2) write(6,*) 'xnhydro_=',xnhydro_(0)
```

```
412:
413:     Fnhydro_imp(0) = ( 1 - xh_imp(0) ) * Fv_out(0) * 0.02
414:     if(pr.ge.2) write(6,*) 'Fnhydro_imp=',Fnhydro_imp(0)
415:
416:     Fhydro_imp_w(0) = w1 * Fcq4_imp(0) + w2 * Fnq3_imp(0) + w3 *
417:    &       Fq2o_imp(0)
418:     if(pr.ge.2) write(6,*) 'Fhydro_imp_w=',Fhydro_imp_w(0)
419:
420:     Ft_pure(0) = Rt_rem(0) * npump - Fhydro_imp_w(0) * it_(0)
421:     if(pr.ge.2) write(6,*) 'Ft_pure=',Ft_pure(0)
422:
423:     Fd_pure(0) = Rd_rem(0) * npump - Fhydro_imp_w(0) * id_(0)
424:     if(pr.ge.2) write(6,*) 'Fd_pure=',Fd_pure(0)
425:
426:     Fp_pure(0) = Rp_rem(0) * npump - Fhydro_imp_w(0) * ip_(0)
427:     if(pr.ge.2) write(6,*) 'Fp_pure=',Fp_pure(0)
428:*
429:* --- Flow into purification ----------------------------------
430:*
431:
432:     Nimpt(0) = Fv_out(0)
433:     if(pr.ge.2) write(6,*) 'Nimpt=',Nimpt(0)
434:
435:     Nnhydro_imp(0) = Fv_out(0) * 0.02 * 0.32
436:     if(pr.ge.2) write(6,*) 'Nnhydro_imp=',Nnhydro_imp(0)
437:
438:     Ncq4_imp(0) = Fv_out(0) * 0.02 * 0.56
439:     if(pr.ge.2) write(6,*) 'Ncq4_imp=',Ncq4_imp(0)
440:
441:     Nnq3_imp(0) = Fv_out(0) * 0.02 * 0.04
442:     if(pr.ge.2) write(6,*) 'Nnq3_imp=',Nnq3_imp(0)
443:
444:     Nq2o_imp(0) = Fv_out(0) * 0.02 * 0.08
445:     if(pr.ge.2) write(6,*) 'Nq2o_imp=',Nq2o_imp(0)
446:
447:     Timp_(0) = Fv_out(0) / 6
448:     if(pr.ge.2) write(6,*) 'Timp_=',Timp_(0)
449:
450:     Dimp_(0) = Fv_out(0) / 6
451:     if(pr.ge.2) write(6,*) 'Dimp_=',Dimp_(0)
452:
453:     Pimp_(0) = Fv_out(0) / 6
454:     if(pr.ge.2) write(6,*) 'Pimp_=',Pimp_(0)
455:
456:     f_hydro_imp_w(0) = w1 * xcq4_ + w2 * xnq3_ + w3 * xq2o_
457:     if(pr.ge.2) write(6,*) 'f_hydro_imp_w=',f_hydro_imp_w(0)
458:
459:     Timp_m(0) = mu_mob * Timp_(0)
460:     if(pr.ge.2) write(6,*) 'Timp_m=',Timp_m(0)
461:
462:     Nhydro_imp(0) = Timp_(0) + Dimp_(0) + Pimp_(0)
463:     if(pr.ge.2) write(6,*) 'Nhydro_imp=',Nhydro_imp(0)
464:
465:     Fh2_puri_in(0) = ( Ft_pure(0) + Fd_pure(0) + Fp_pure(0) ) / 2
466:     if(pr.ge.2) write(6,*) 'Fh2_puri_in=',Fh2_puri_in(0)
467:
468:     FF = Ft_pure(0) + Fd_pure(0) + Fp_pure(0)
469:
470:     if ( FF .gt. 0.0d0 ) then
471:
472:     Ft2_sub_in(0) = Ft_pure(0) * ( 0.5 + ( xnhydro_(0) + xh_imp(0) )
473:    &       / FF * Fadj_puri(0) * Lhydro_ / Limp_ )
474:     if(pr.ge.2) write(6,*) 'Ft2_sub_in=',Ft2_sub_in(0)
475:
476:     Fd2_sub_in(0) = Fd_pure(0) * ( 0.5 + ( xnhydro_(0) + xh_imp(0) )
477:    &       / FF * Fadj_puri(0) * Lhydro_ / Limp_ )
478:     if(pr.ge.2) write(6,*) 'Fd2_sub_in=',Fd2_sub_in(0)
479:
480:     Fp2_sub_in(0) = Fp_pure(0) * ( 0.5 + ( xnhydro_(0) + xh_imp(0) )
481:    &       / FF * Fadj_puri(0) * Lhydro_ / Limp_ )
482:     if(pr.ge.2) write(6,*) 'Fp2_sub_in=',Fp2_sub_in(0)
483:
484:     else
485:
486:     Ft2_sub_in(0) = 0.0
487:     if(pr.ge.2) write(6,*) 'Ft2_sub_in=',Ft2_sub_in(0)
488:
489:     Fd2_sub_in(0) = 0.0
490:     if(pr.ge.2) write(6,*) 'Fd2_sub_in=',Fd2_sub_in(0)
491:
492:     Fp2_sub_in(0) = 0.0
493:     if(pr.ge.2) write(6,*) 'Fp2_sub_in=',Fp2_sub_in(0)
494:
495:     endif
496:
497:     Fh2_sub_in(0) = Ft2_sub_in(0) + Fd2_sub_in(0) + Fp2_sub_in(0)
498:     if(pr.ge.2) write(6,*) 'Fh2_sub_in=',Fh2_sub_in(0)
499:
500:     if ( Fp2_sub_in(0) .gt. 1.0d-10 ) then
501:
502:     xt_sub(0) = Ft2_sub_in(0) / Fh2_sub_in(0)
503:     if(pr.ge.2) write(6,*) 'xt_sub=',xt_sub(0)
504:
505:     xd_sub(0) = Fd2_sub_in(0) / Fh2_sub_in(0)
506:     if(pr.ge.2) write(6,*) 'xd_sub=',xd_sub(0)
507:
508:     xp_sub(0) = Fp2_sub_in(0) / Fh2_sub_in(0)
509:     if(pr.ge.2) write(6,*) 'xp_sub=',xp_sub(0)
510:
511:     else
512:
513:     xt_sub(0) = 0.0d0
514:     if(pr.ge.2) write(6,*) 'xt_sub=',xt_sub(0)
515:
516:     xd_sub(0) = 0.0d0
517:     if(pr.ge.2) write(6,*) 'xd_sub=',xd_sub(0)
518:
519:     xp_sub(0) = 0.0d0
520:     if(pr.ge.2) write(6,*) 'xp_sub=',xp_sub(0)
521:
522:     endif
523:
524:     if ( abs ( Fh2_sub_in(0) ) .lt. 1.0d-10 ) then
525:
```

```
526:     tau_sb = 0.0d0
527:
528:     else
529:
530:     tau_sb = Ns_max / Fh2_sub_in(0)
531:
532:     endif
533:
534:     Ns(0) = Ns_max
535:     if(pr.ge.2) write(6,*) 'Ns=',Ns(0)
536:
537:     T2_sub(0) = Ns_max / 3
538:     if(pr.ge.2) write(6,*) 'T2_sub=',T2_sub(0)
539:
540:     D2_sub(0) = Ns_max / 3
541:     if(pr.ge.2) write(6,*) 'D2_sub=',D2_sub(0)
542:
543:     P2_sub(0) = Ns_max / 3
544:     if(pr.ge.2) write(6,*) 'P2_sub=',P2_sub(0)
545:
546:     Nhydro_sub(0) = T2_sub(0) + D2_sub(0) + P2_sub(0)
547:     if(pr.ge.2) write(6,*) 'Nhydro_sub=',Nhydro_sub(0)
548:
549:     if ( abs ( tau_sb ) .ge. 1.0d-10 ) then
550:
551:     Ft_sub_out(0) = 2.0 * ( Ft2_sub_in(0) + xt_sub(0) * Ns(0) /
552:    &       tau_sb )
553:     if(pr.ge.2) write(6,*) 'Ft_sub_out=',Ft_sub_out(0)
554:
555:     Fd_sub_out(0) = 2.0 * ( Fd2_sub_in(0) + xd_sub(0) * Ns(0) /
556:    &       tau_sb )
557:     if(pr.ge.2) write(6,*) 'Fd_sub_out=',Fd_sub_out(0)
558:
559:     Fp_sub_out(0) = 2.0 * ( Fp2_sub_in(0) + xp_sub(0) * Ns(0) /
560:    &       tau_sb )
561:     if(pr.ge.2) write(6,*) 'Fp_sub_out=',Fp_sub_out(0)
562:
563:     else
564:
565:     Ft_sub_out(0) = 0.0d0
566:     if(pr.ge.2) write(6,*) 'Ft_sub_out=',Ft_sub_out(0)
567:
568:     Fd_sub_out(0) = 0.0d0
569:     if(pr.ge.2) write(6,*) 'Fd_sub_out=',Fd_sub_out(0)
570:
571:     Fp_sub_out(0) = 0.0d0
572:     if(pr.ge.2) write(6,*) 'Fp_sub_out=',Fp_sub_out(0)
573:
574:     endif
575:
576:     Fh_sub_out(0) = Ft_sub_out(0) + Fd_sub_out(0) + Fp_sub_out(0)
577:     if(pr.ge.2) write(6,*) 'Fh_sub_out=',Fh_sub_out(0)
578:
579:c    if ( tpoint_cycle .lt. treg_msieve ) then
580:
581:     Treg_out(0) = Timp_(0) * ( 1 - tpoint_cycle / treg_msieve )
582:     if(pr.ge.2) write(6,*) 'Treg_out=',Treg_out(0)
583:
584:c    else
585:
586:c    Treg_out(0) = 0D0
587:c    if(pr.ge.2) write(6,*) 'Treg_out=',Treg_out(0)
588:
589:c    endif
590:
591:     Treg_out_m(0) = mu_reg_m * Treg_out(0)
592:     if(pr.ge.2) write(6,*) 'Treg_out_m=',Treg_out_m(0)
593:* Evaluation of Hydrogen-containing Species:
594:
595:     Ncq4_reg(0) = Ncq4_imp(j)
596:     if(pr.ge.2) write(6,*) 'Ncq4_reg=',Ncq4_reg(0)
597:
598:     Nnq3_reg(0) = Nnq3_imp(j)
599:     if(pr.ge.2) write(6,*) 'Nnq3_reg=',Nnq3_reg(0)
600:
601:     Nq2o_reg(0) = Nq2o_imp(j)
602:     if(pr.ge.2) write(6,*) 'Nq2o_reg=',Nq2o_reg(0)
603:
604:     Ecq4_reg(0) = Ncq4_reg(0) / treg_msieve
605:     if(pr.ge.2) write(6,*) 'Ecq4_reg=',Ecq4_reg(0)
606:
607:     Enq3_reg(0) = Nnq3_reg(0) / treg_msieve
608:     if(pr.ge.2) write(6,*) 'Enq3_reg=',Enq3_reg(0)
609:
610:     Eq2o_reg(0) = Nq2o_reg(0) / treg_msieve
611:     if(pr.ge.2) write(6,*) 'Eq2o_reg=',Eq2o_reg(0)
612:
613:     Fhydro_pmem(0) = w1 * Ecq4_reg(0) + w2 * Enq3_reg(0) + w3 *
614:    &       Eq2o_reg(0)
615:     if(pr.ge.2) write(6,*) 'Fhydro_pmem=',Fhydro_pmem(0)
616:
617:     if ( Nhydro_imp(j) .ne. 0.0d0 ) then
618:
619:     it_endj(0) = Timp_(j) / Nhydro_imp(j)
620:     if(pr.ge.2) write(6,*) 'it_endj=',it_endj(0)
621:
622:     id_endj(0) = Pimp_(j) / Nhydro_imp(j)
623:     if(pr.ge.2) write(6,*) 'id_endj=',id_endj(0)
624:
625:     ip_endj(0) = Timp_(j) / Nhydro_imp(j)
626:     if(pr.ge.2) write(6,*) 'ip_endj=',ip_endj(0)
627:
628:     else
629:
630:     it_endj(0) = 0.0D0
631:     if(pr.ge.2) write(6,*) 'it_endj=',it_endj(0)
632:
633:     id_endj(0) = 0.0D0
634:     if(pr.ge.2) write(6,*) 'id_endj=',id_endj(0)
635:
636:     ip_endj(0) = 0.0D0
637:     if(pr.ge.2) write(6,*) 'ip_endj=',ip_endj(0)
638:
639:     endif
```

```
640:
641:     Tpmem(0) = 0.0D0
642:     if(pr.ge.2) write(6,*) 'Tpmem=',Tpmem(0)
643:
644:     Dpmem(0) = 0.0D0
645:     if(pr.ge.2) write(6,*) 'Dpmem=',Dpmem(0)
646:
647:     Ppmem(0) = 0.0D0
648:     if(pr.ge.2) write(6,*) 'Ppmem=',Ppmem(0)
649:
650:     Ft_puri_out(0) = Ft_sub_out(0) + Tpmem(0) / tau_pmem
651:     if(pr.ge.2) write(6,*) 'Ft_puri_out=',Ft_puri_out(0)
652:
653:     Fd_puri_out(0) = Fd_sub_out(0) + Dpmem(0) / tau_pmem
654:     if(pr.ge.2) write(6,*) 'Fd_puri_out=',Fd_puri_out(0)
655:
656:     Fp_puri_out(0) = Fp_sub_out(0) + Ppmem(0) / tau_pmem
657:     if(pr.ge.2) write(6,*) 'Fp_puri_out=',Fp_puri_out(0)
658:
659:     Fpuri_out(0) = Ft_puri_out(0) + Fd_puri_out(0) + Fp_puri_out(0)
660:     if(pr.ge.2) write(6,*) 'Fpuri_out=',Fpuri_out(0)
661:*
662:* ... Fiss_in: assuming Fragnb = Fragss_inj = 0
663:
664:     Fiss_in(0) = Fbl_cool + Fbl_ + ( 1 - h1 - h3 ) * Fpuri_out(0)
665:     if(pr.ge.2) write(6,*) 'Fiss_in=',Fiss_in(0)
666:
667:     Niss_(0) = Fiss_in(0)
668:     if(pr.ge.2) write(6,*) 'Niss_=',Niss_(0)
669:*Tiss_   = Niss_ / 3.0D0 ;
670:*Diss_   = Niss_ / 3.0D0 ;
671:
672:     Tiss_(0) = Niss_(0) * 3.0 / ( 3.0 + 3.0 + 1.0 )
673:     if(pr.ge.2) write(6,*) 'Tiss_=',Tiss_(0)
674:
675:     Diss_(0) = Niss_(0) * 3.0 / ( 3.0 + 3.0 + 1.0 )
676:     if(pr.ge.2) write(6,*) 'Diss_=',Diss_(0)
677:
678:     if ( Niss_(0) .gt. 0.0 ) then
679:
680:     Xiss_t(0) = Tiss_(0) / Niss_(0)
681:     if(pr.ge.2) write(6,*) 'Xiss_t=',Xiss_t(0)
682:
683:     Xiss_d(0) = Diss_(0) / Niss_(0)
684:     if(pr.ge.2) write(6,*) 'Xiss_d=',Xiss_d(0)
685:
686:     Xiss_p(0) = 1.0d0 - Xiss_t(0) - Xiss_d(0)
687:     if(pr.ge.2) write(6,*) 'Xiss_p=',Xiss_p(0)
688:
689:     else
690:
691:     Xiss_t(0) = 0.0d0
692:     if(pr.ge.2) write(6,*) 'Xiss_t=',Xiss_t(0)
693:
694:     Xiss_d(0) = 0.0d0
695:     if(pr.ge.2) write(6,*) 'Xiss_d=',Xiss_d(0)
696:
697:     Xiss_p(0) = 0.0d0
698:     if(pr.ge.2) write(6,*) 'Xiss_p=',Xiss_p(0)
699:
700:     endif
701:**Piss_   = Niss_ / 3.0D0 ;
702:
703:     Piss_(0) = Xiss_p(0) * Niss_(0)
704:     if(pr.ge.2) write(6,*) 'Piss_=',Piss_(0)
705:
706:     Tiss_m(0) = mu_iss_ * Tiss_(0)
707:     if(pr.ge.2) write(6,*) 'Tiss_m=',Tiss_m(0)
708:
709:     P1 = 1.0 / ( Xiss_mid_t * Xiss_bot_d + Xiss_bot_t * Xiss_top_d +
710:    &      Xiss_top_t * Xiss_mid_d - Xiss_mid_t * Xiss_top_d -
711:    &      Xiss_top_t * Xiss_bot_d - Xiss_bot_t * Xiss_mid_d )
712:c============================
713:
714:     Fiss_top_(0) = P1 * ( ( Xiss_mid_t * Xiss_bot_d - Xiss_bot_t *
715:    &      Xiss_mid_d ) * Niss_(0) / tau_iss_ + ( Xiss_mid_d -
716:    &      Xiss_bot_d ) * Tiss_(0) / tau_iss_ + ( Xiss_bot_t -
717:    &      Xiss_mid_t ) * Diss_(0) / tau_iss_ )
718:     if(pr.ge.2) write(6,*) 'Fiss_top_=',Fiss_top_(0)
719:
720:     Fiss_mid_(0) = P1 * ( ( Xiss_bot_t * Xiss_top_d - Xiss_top_t *
721:    &      Xiss_bot_d ) * Niss_(0) / tau_iss_ + ( Xiss_bot_d -
722:    &      Xiss_top_d ) * Tiss_(0) / tau_iss_ + ( Xiss_top_t -
723:    &      Xiss_bot_t ) * Diss_(0) / tau_iss_ )
724:     if(pr.ge.2) write(6,*) 'Fiss_mid_=',Fiss_mid_(0)
725:
726:     Fiss_bot_(0) = P1 * ( ( Xiss_top_t * Xiss_mid_d - Xiss_mid_t *
727:    &      Xiss_top_d ) * Niss_(0) / tau_iss_ + ( Xiss_top_d -
728:    &      Xiss_mid_d ) * Tiss_(0) / tau_iss_ + ( Xiss_mid_t -
729:    &      Xiss_top_t ) * Diss_(0) / tau_iss_ )
730:     if(pr.ge.2) write(6,*) 'Fiss_bot_=',Fiss_bot_(0)
731:*
732:* --- Pellet injector ----------------------------------------
733:*
734:
735:     Fsto_out_t(0) = Fsto_tri_out(0) / 2
736:     if(pr.ge.2) write(6,*) 'Fsto_out_t=',Fsto_out_t(0)
737:
738:     Fsto_out_d(0) = Fsto_deu_out(0) / 4
739:     if(pr.ge.2) write(6,*) 'Fsto_out_d=',Fsto_out_d(0)
740:
741:     Next_12(0) = Fplas_in_t(0) / 2
742:     if(pr.ge.2) write(6,*) 'Next_12=',Next_12(0)
743:
744:     Text_12(0) = Next_12(0) / 3
745:     if(pr.ge.2) write(6,*) 'Text_12=',Text_12(0)
746:
747:     Dext_12(0) = Next_12(0) / 3
748:     if(pr.ge.2) write(6,*) 'Dext_12=',Dext_12(0)
749:
750:     Pext_12(0) = Next_12(0) / 3
751:     if(pr.ge.2) write(6,*) 'Pext_12=',Pext_12(0)
752:
753:     Xext_12_t(0) = Text_12(0) / Next_12(0)
```

```
754:     if(pr.ge.2) write(6,*) 'Xext_12_t=',Xext_12_t(0)
755:
756:     Xext_12_d(0) = Dext_12(0) / Next_12(0)
757:     if(pr.ge.2) write(6,*) 'Xext_12_d=',Xext_12_d(0)
758:
759:     Xext_12_p(0) = 1 - Xext_12_t(0) - Xext_12_d(0)
760:     if(pr.ge.2) write(6,*) 'Xext_12_p=',Xext_12_p(0)
761:
762:     Next_tot(0) = Next_12(0) * 2.0
763:     if(pr.ge.2) write(6,*) 'Next_tot=',Next_tot(0)
764:
765:     Text_tot(0) = Text_12(0) * 2.0
766:     if(pr.ge.2) write(6,*) 'Text_tot=',Text_tot(0)
767:
768:     Dext_tot(0) = Dext_12(0) * 2.0
769:     if(pr.ge.2) write(6,*) 'Dext_tot=',Dext_tot(0)
770:
771:     Pext_tot(0) = Pext_12(0) * 2.0
772:     if(pr.ge.2) write(6,*) 'Pext_tot=',Pext_tot(0)
773:
774:     Text_m(0) = mu_ext * Text_tot(0)
775:     if(pr.ge.2) write(6,*) 'Text_m=',Text_m(0)
776:
777:     Nextru_(0) = Next_12(0)
778:     if(pr.ge.2) write(6,*) 'Nextru_=',Nextru_(0)
779:
780:     Xextru_t(0) = Text_12(0) / Next_12(0)
781:     if(pr.ge.2) write(6,*) 'Xextru_t=',Xextru_t(0)
782:
783:     Xextru_d(0) = Dext_12(0) / Next_12(0)
784:     if(pr.ge.2) write(6,*) 'Xextru_d=',Xextru_d(0)
785:
786:     Xextru_p(0) = 1.0d0 - Xextru_t(0) - Xextru_d(0)
787:     if(pr.ge.2) write(6,*) 'Xextru_p=',Xextru_p(0)
788:
789:     Vinj_t(0) = Xextru_t(0) * Nextru_(0) * 18.894D-6 / 2
790:     if(pr.ge.2) write(6,*) 'Vinj_t=',Vinj_t(0)
791:
792:     Vinj_d(0) = Xextru_d(0) * Nextru_(0) * 20.085D-6 / 2
793:     if(pr.ge.2) write(6,*) 'Vinj_d=',Vinj_d(0)
794:
795:     Vinj_p(0) = Xextru_p(0) * Nextru_(0) * 23.326D-6 / 2
796:     if(pr.ge.2) write(6,*) 'Vinj_p=',Vinj_p(0)
797:* total volume in extruder
798:
799:     Vinj_tot(0) = Vinj_t(0) + Vinj_d(0) + Vinj_p(0)
800:     if(pr.ge.2) write(6,*) 'Vinj_tot=',Vinj_tot(0)
801:
802:     npellet_t(0) = Vinj_t(0) / Vinj_tot(0) * Vpellet * 2 * kpellet_t
803:     if(pr.ge.2) write(6,*) 'npellet_t=',npellet_t(0)
804:
805:     npellet_d(0) = Vinj_d(0) / Vinj_tot(0) * Vpellet * 2 * kpellet_d
806:     if(pr.ge.2) write(6,*) 'npellet_d=',npellet_d(0)
807:
808:     npellet_p(0) = Vinj_p(0) / Vinj_tot(0) * Vpellet * 2 * kpellet_p
809:     if(pr.ge.2) write(6,*) 'npellet_p=',npellet_p(0)
810:
811:     Fbk_plas_(0) = 3 * 5.98D-22 * nedge_tot
812:     if(pr.ge.2) write(6,*) 'Fbk_plas_=',Fbk_plas_(0)
813:
814:     Fbk_plas_t(0) = 3 * 5.98D-22 * nedge_t
815:     if(pr.ge.2) write(6,*) 'Fbk_plas_t=',Fbk_plas_t(0)
816:
817:     Fbk_plas_d(0) = 3 * 5.98D-22 * nedge_d
818:     if(pr.ge.2) write(6,*) 'Fbk_plas_d=',Fbk_plas_d(0)
819:
820:     Fbk_plas_p(0) = 3 * 5.98D-22 * nedge_p
821:     if(pr.ge.2) write(6,*) 'Fbk_plas_p=',Fbk_plas_p(0)
822:
823:     Finj_prop(0) = Fsto_out_p(0) + liss_p * Fiss_top_(0)
824:     if(pr.ge.2) write(6,*) 'Finj_prop=',Finj_prop(0)
825:
826:     Rinj_(0) = Finj_prop(0) / nprop
827:     if(pr.ge.2) write(6,*) 'Rinj_=',Rinj_(0)
828:
829:     Fplas_in_p(0) = Rinj_(0) * npellet_p(0) * ( 1 - finj_err )
830:     if(pr.ge.2) write(6,*) 'Fplas_in_p=',Fplas_in_p(0)
831:
832:     Fsto_pro_out_p(0) = Fsto_out_p(0) * Xsto_pro_p(0)
833:     if(pr.ge.2) write(6,*) 'Fsto_pro_out_p=',Fsto_pro_out_p(0)
834:
835:     Fplas_prop(0) = ( 1 - fcap_prop ) * Finj_prop(0)
836:     if(pr.ge.2) write(6,*) 'Fplas_prop=',Fplas_prop(0)
837:*
838:*Fuel_p = Fplas_in_p + Xgas_p * Fgas_tot + Fplas_prop_p ;
839:*Fex_p = Reac_p + Fuel_p;
840:*
841:
842:     Xprop_t(0) = ( Fsto_out_p(0) * Xsto_pro_t(0) + liss_p *
843:    &      Fiss_top_(0) * Xiss_top_t ) / Finj_prop(0)
844:     if(pr.ge.2) write(6,*) 'Xprop_t=',Xprop_t(0)
845:
846:     Xprop_d(0) = ( Fsto_out_p(0) * Xsto_pro_d(0) + liss_p *
847:    &      Fiss_top_(0) * Xiss_top_d ) / Finj_prop(0)
848:     if(pr.ge.2) write(6,*) 'Xprop_d=',Xprop_d(0)
849:
850:     Xprop_p(0) = 1 - Xprop_t(0) - Xprop_d(0)
851:     if(pr.ge.2) write(6,*) 'Xprop_p=',Xprop_p(0)
852:
853:     Fplas_prop_t(0) = Xprop_t(0) * Fplas_prop(0)
854:     if(pr.ge.2) write(6,*) 'Fplas_prop_t=',Fplas_prop_t(0)
855:
856:     Fplas_prop_d(0) = Xprop_d(0) * Fplas_prop(0)
857:     if(pr.ge.2) write(6,*) 'Fplas_prop_d=',Fplas_prop_d(0)
858:
859:     Fplas_prop_p(0) = Xprop_p(0) * Fplas_prop(0)
860:     if(pr.ge.2) write(6,*) 'Fplas_prop_p=',Fplas_prop_p(0)
861:
862:     Finj_in_tot(0) = Fplas_in_t(0) + Fplas_in_d(0) + Fplas_in_p(0)
863:     if(pr.ge.2) write(6,*) 'Finj_in_tot=',Finj_in_tot(0)
864:
865:     Finj_exh(0) = ( ( Fplas_in_t(0) + Fplas_in_d(0) ) / ( 1 -
866:    &      finj_err ) + Fplas_in_p(0) ) * finj_err + Finj_prop(0) *
867:    &      fcap_prop + Fbk_plas_(0)
```

page 29

```
868:     if(pr.ge.2) write(6,*) 'Finj_exh=',Finj_exh(0)
869:
870:     Finj_exh_t(0) = Fplas_in_t(0) * finj_err / ( 1 - finj_err ) +
871:    &      Finj_prop(0) * Xprop_t(0) * fcap_prop + Fbk_plas_t(0)
872:     if(pr.ge.2) write(6,*) 'Finj_exh_t=',Finj_exh_t(0)
873:
874:     Finj_exh_d(0) = Fplas_in_d(0) * finj_err / ( 1 - finj_err ) +
875:    &      Finj_prop(0) * Xprop_d(0) * fcap_prop + Fbk_plas_d(0)
876:     if(pr.ge.2) write(6,*) 'Finj_exh_d=',Finj_exh_d(0)
877:
878:     Fiss_inj(0) = ( 1 - fnon_iss ) * Finj_exh(0)
879:     if(pr.ge.2) write(6,*) 'Fiss_inj=',Fiss_inj(0)
880:
881:     Finj_rec(0) = fnon_iss * Finj_exh(0)
882:     if(pr.ge.2) write(6,*) 'Finj_rec=',Finj_rec(0)
883:
884:     Xinj_t(0) = Finj_exh_t(0) / Finj_exh(0)
885:     if(pr.ge.2) write(6,*) 'Xinj_t=',Xinj_t(0)
886:
887:     Xinj_d(0) = Finj_exh_d(0) / Finj_exh(0)
888:     if(pr.ge.2) write(6,*) 'Xinj_d=',Xinj_d(0)
889:
890:     Xinj_p(0) = 1 - Xinj_t(0) - Xinj_d(0)
891:     if(pr.ge.2) write(6,*) 'Xinj_p=',Xinj_p(0)
892:
893:     Fgas_in_tri(0) = Fsto_tri_out(0) / 2
894:     if(pr.ge.2) write(6,*) 'Fgas_in_tri=',Fgas_in_tri(0)
895:*Fgas_d = Fgas_in_tri(t) * Xsto_tri_d + Fgas_in_deu(t) * Xsto_deu_d
896:*     + h1 * Fpuri_out(t0) * Xiss_in_d
897:*     + mgas_mid_iss * Fiss_mid_(t0) * Xiss_mid_d
898:*     + mgas_bot_iss * Fiss_bot_(t0) * Xiss_bot_d ;
899:
900:     Ngas_tot(0) = Burn_t(0) / 6.0
901:     if(pr.ge.2) write(6,*) 'Ngas_tot=',Ngas_tot(0)
902:
903:     Tgas_(0) = Burn_t(0) / 18.0
904:     if(pr.ge.2) write(6,*) 'Tgas_=',Tgas_(0)
905:
906:     Tgas_mob(0) = mu_gas * Tgas_(0)
907:     if(pr.ge.2) write(6,*) 'Tgas_mob=',Tgas_mob(0)
908:
909:     Dgas_(0) = Burn_t(0) / 18.0
910:     if(pr.ge.2) write(6,*) 'Dgas_=',Dgas_(0)
911:
912:     Xgas_t(0) = Tgas_(0) / Ngas_tot(0)
913:     if(pr.ge.2) write(6,*) 'Xgas_t=',Xgas_t(0)
914:
915:     Xgas_d(0) = Dgas_(0) / Ngas_tot(0)
916:     if(pr.ge.2) write(6,*) 'Xgas_d=',Xgas_d(0)
917:
918:     Xgas_p(0) = 1 - Xgas_t(0) - Xgas_d(0)
919:     if(pr.ge.2) write(6,*) 'Xgas_p=',Xgas_p(0)
920:
921:     Pgas_(0) = Xgas_p(0) * Ngas_tot(0)
922:     if(pr.ge.2) write(6,*) 'Pgas_=',Pgas_(0)
923:*Fgas_tot = (Fgas_t(t)+Fgas_d(t)) / (Xgas_t(t)+Xgas_d(t)) ;
924:*Fpuff_plas = Fgas_tot + Fplas_prop_t + Fplas_prop_d + Fplas_prop_p ;
925:*Fgas_d = Fgas_in_tri(t) * Xsto_tri_d + Fgas_in_deu(t) * Xsto_deu_d
926:*     + h1 * Fpuri_out(t0) * Xiss_in_d
927:*     + mgas_mid_iss * Fiss_mid_(t0) * Xiss_mid_d
928:*     + mgas_bot_iss * Fiss_bot_(t0) * Xiss_bot_d ;
929:*Fgas_tot = (Fgas_t(t)+Fgas_d(t)) / (Xgas_t(t)+Xgas_d(t)) ;
930:*Fpuff_plas = Fgas_tot + Fplas_prop_t + Fplas_prop_d + Fplas_prop_p ;
931:*
932:*  --- Isotope Separation System ------------------------------
933:*
934:*
935:*
936:
937:     Fiss_in_t(0) = Fbl_cool * Xc_t + Fbl_ * Xbl_t
938:     if(pr.ge.2) write(6,*) 'Fiss_in_t=',Fiss_in_t(0)
939:
940:     Fiss_in_d(0) = Fbl_cool * Xc_d + Fbl_ * Xbl_d
941:     if(pr.ge.2) write(6,*) 'Fiss_in_d=',Fiss_in_d(0)
942:
943:     Fiss_in_p(0) = Fiss_in(0) - Fiss_in_t(0) - Fiss_in_d(0)
944:     if(pr.ge.2) write(6,*) 'Fiss_in_p=',Fiss_in_p(0)
945:
946:     if ( Fiss_in(0) .ne. 0.0d0 ) then
947:
948:     Xiss_in_t(0) = Fiss_in_t(0) / Fiss_in(0)
949:     if(pr.ge.2) write(6,*) 'Xiss_in_t=',Xiss_in_t(0)
950:
951:     Xiss_in_d(0) = Fiss_in_d(0) / Fiss_in(0)
952:     if(pr.ge.2) write(6,*) 'Xiss_in_d=',Xiss_in_d(0)
953:
954:     Xiss_in_p(0) = 1 - Xiss_in_t(0) - Xiss_in_d(0)
955:     if(pr.ge.2) write(6,*) 'Xiss_in_p=',Xiss_in_p(0)
956:
957:     else
958:
959:     Xiss_in_t(0) = 0.0d0
960:     if(pr.ge.2) write(6,*) 'Xiss_in_t=',Xiss_in_t(0)
961:
962:     Xiss_in_d(0) = 0.0d0
963:     if(pr.ge.2) write(6,*) 'Xiss_in_d=',Xiss_in_d(0)
964:
965:     Xiss_in_p(0) = 0.0d0
966:     if(pr.ge.2) write(6,*) 'Xiss_in_p=',Xiss_in_p(0)
967:
968:     endif
969:* --- Fgas_d / Fgas_tot / Fpuff_plas are here bacause Fgas_d use
970:*    Xiss_in_d
971:
972:     Fgas_d(0) = Fgas_in_tri(0) * Xsto_tri_d(0) + Fgas_in_deu(0) *
973:    &      Xsto_deu_d(0) + h1 * Fpuri_out(0) * Xiss_in_d(0) +
974:    &      mgas_mid_iss * Fiss_mid_(0) * Xiss_mid_d + mgas_bot_iss *
975:    &      Fiss_bot_(0) * Xiss_bot_d
976:     if(pr.ge.2) write(6,*) 'Fgas_d=',Fgas_d(0)
977:
978:     Fgas_tot(0) = ( Fgas_t(0) + Fgas_d(0) ) / ( Xgas_t(0) + Xgas_d(0)
979:    &      )
980:     if(pr.ge.2) write(6,*) 'Fgas_tot=',Fgas_tot(0)
981:
```

```
 982:     Fpuff_plas(0) = Fgas_tot(0) + Fplas_prop_t(0) + Fplas_prop_d(0) +
 983:    &      Fplas_prop_p(0)
 984:     if(pr.ge.2) write(6,*) 'Fpuff_plas=',Fpuff_plas(0)
 985:*
 986:*  --- Neutral beam injector ------------------------------
 987:*
 988:
 989:     W_nb(0) = 75D0 * fppower0 ( 0.0d0 , t_burn , t_p_ru , t_p_rd
 990:    &                          , t_dwell )
 991:     if(pr.ge.2) write(6,*) 'W_nb=',W_nb(0)
 992:
 993:     Fplas_nb_d(0) = 3.6D3 / 1.6D0 * 1.0D19 * 1 / No * W_nb(0) / E_nb
 994:     if(pr.ge.2) write(6,*) 'Fplas_nb_d=',Fplas_nb_d(0)
 995:**Fin_st_d    = Fsto_deu_out / 2.0D0 ;
 996:
 997:     Fin_st_d(0) = 0.0D0
 998:     if(pr.ge.2) write(6,*) 'Fin_st_d=',Fin_st_d(0)
 999:
1000:     Ncryo_(0) = Burn_d(0) / 2.0D0
1001:     if(pr.ge.2) write(6,*) 'Ncryo_=',Ncryo_(0)
1002:
1003:     Tnb_cryo(0) = Ncryo_(0) / 3.0D0
1004:     if(pr.ge.2) write(6,*) 'Tnb_cryo=',Tnb_cryo(0)
1005:
1006:     Dnb_cryo(0) = Ncryo_(0) / 3.0D0
1007:     if(pr.ge.2) write(6,*) 'Dnb_cryo=',Dnb_cryo(0)
1008:
1009:     Xnb_cryo_t(0) = Tnb_cryo(0) / Ncryo_(0)
1010:     if(pr.ge.2) write(6,*) 'Xnb_cryo_t=',Xnb_cryo_t(0)
1011:
1012:     Xnb_cryo_d(0) = Dnb_cryo(0) / Ncryo_(0)
1013:     if(pr.ge.2) write(6,*) 'Xnb_cryo_d=',Xnb_cryo_d(0)
1014:
1015:     Xnb_cryo_p(0) = 1.0 - Xnb_cryo_d(0) - Xnb_cryo_t(0)
1016:     if(pr.ge.2) write(6,*) 'Xnb_cryo_p=',Xnb_cryo_p(0)
1017:
1018:     Pnb_cryo(0) = Xnb_cryo_p(0) * Ncryo_(0)
1019:     if(pr.ge.2) write(6,*) 'Pnb_cryo=',Pnb_cryo(0)
1020:
1021:     Rnb_reg_p(0) = Pnb_cryo(0) * emrRsAp
1022:     if(pr.ge.2) write(6,*) 'Rnb_reg_p=',Rnb_reg_p(0)
1023:
1024:     tnb_(0) = t - nnb_ * tnb_pump
1025:     if(pr.ge.2) write(6,*) 'tnb_=',tnb_(0)
1026:
1027:     if ( tnb_(0) .gt. tnb_reg .and. tnb_(0) .lt. tnb_pump ) then
1028:
1029:     Rnb_reg_t(0) = 0.0d0
1030:     if(pr.ge.2) write(6,*) 'Rnb_reg_t=',Rnb_reg_t(0)
1031:
1032:     Rnb_reg_d(0) = 0.0d0
1033:     if(pr.ge.2) write(6,*) 'Rnb_reg_d=',Rnb_reg_d(0)
1034:
1035:     else
1036:
1037:     Rnb_reg_t(0) = Tnb_cryo(0) * emrRsAp
1038:     if(pr.ge.2) write(6,*) 'Rnb_reg_t=',Rnb_reg_t(0)
1039:
1040:     Rnb_reg_d(0) = Dnb_cryo(0) * emrRsAp
1041:     if(pr.ge.2) write(6,*) 'Rnb_reg_d=',Rnb_reg_d(0)
1042:
1043:     endif
1044:
1045:     Rnb_reg_(0) = Ncryo_(0) * emrRsAp
1046:     if(pr.ge.2) write(6,*) 'Rnb_reg_=',Rnb_reg_(0)
1047:
1048:     Fnb_cryo(0) = Rnb_reg_t(0) + Rnb_reg_d(0) + Rnb_reg_p(0)
1049:     if(pr.ge.2) write(6,*) 'Fnb_cryo=',Fnb_cryo(0)
1050:
1051:     Crec(0) = 0.0D0
1052:     if(pr.ge.2) write(6,*) 'Crec=',Crec(0)
1053:
1054:     Fin_rec_(0) = Crec(0) * Fnb_cryo(0)
1055:     if(pr.ge.2) write(6,*) 'Fin_rec_=',Fin_rec_(0)
1056:
1057:     Fiss_nb(0) = ( 1.0D0 - Crec(0) ) * Fnb_cryo(0)
1058:     if(pr.ge.2) write(6,*) 'Fiss_nb=',Fiss_nb(0)
1059:
1060:     if ( tnb_(0) .gt. 0.0d0 .and. tnb_(0) .lt. tnb_reg ) then
1061:
1062:     Tnb_reg_(0) = Rnb_reg_t(0) * tnb_reg
1063:     if(pr.ge.2) write(6,*) 'Tnb_reg_=',Tnb_reg_(0)
1064:
1065:     Dnb_reg_(0) = Rnb_reg_d(0) * tnb_reg
1066:     if(pr.ge.2) write(6,*) 'Dnb_reg_=',Dnb_reg_(0)
1067:
1068:     Dnb_cryo_tot(0) = Dnb_cryo(0) + Dnb_reg_(0) * ( 1.0 - tnb_(0) /
1069:    &       tnb_reg )
1070:     if(pr.ge.2) write(6,*) 'Dnb_cryo_tot=',Dnb_cryo_tot(0)
1071:
1072:     elseif ( tnb_(0) .gt. tnb_reg .and. tnb_(0) .lt. tnb_pump ) then
1073:
1074:     Tnb_reg_(0) = 0.0d0
1075:     if(pr.ge.2) write(6,*) 'Tnb_reg_=',Tnb_reg_(0)
1076:
1077:     Dnb_reg_(0) = 0.0d0
1078:     if(pr.ge.2) write(6,*) 'Dnb_reg_=',Dnb_reg_(0)
1079:
1080:     Dnb_cryo_tot(0) = Dnb_cryo(0)
1081:     if(pr.ge.2) write(6,*) 'Dnb_cryo_tot=',Dnb_cryo_tot(0)
1082:
1083:     endif
1084:
1085:     if ( tnb_(0) .lt. tnb_reg ) then
1086:
1087:     Tnb_cryo_tot(0) = Tnb_cryo(0) + Tnb_reg_(0) * ( 1.0 - tnb_(0) /
1088:    &       tnb_reg )
1089:     if(pr.ge.2) write(6,*) 'Tnb_cryo_tot=',Tnb_cryo_tot(0)
1090:
1091:     elseif ( tnb_(0) .gt. tnb_reg .and. tnb_(0) .lt. tnb_pump ) then
1092:
1093:     Tnb_cryo_tot(0) = Tnb_cryo(0)
1094:     if(pr.ge.2) write(6,*) 'Tnb_cryo_tot=',Tnb_cryo_tot(0)
1095:
```

page 31

```
1096:     endif
1097:
1098:     Tnb_cryo_mob(0) = mu_nb_mob * Tnb_cryo_tot(0)
1099:     if(pr.ge.2) write(6,*) 'Tnb_cryo_mob=',Tnb_cryo_mob(0)
1100:
1101:     Fbk_nb_(0) = 3 * 5.98D-21 * nedge_tot
1102:     if(pr.ge.2) write(6,*) 'Fbk_nb_=',Fbk_nb_(0)
1103:
1104:     Fbk_nb_t(0) = 3 * 5.98D-21 * nedge_t
1105:     if(pr.ge.2) write(6,*) 'Fbk_nb_t=',Fbk_nb_t(0)
1106:
1107:     Fbk_nb_d(0) = 3 * 5.98D-21 * nedge_d
1108:     if(pr.ge.2) write(6,*) 'Fbk_nb_d=',Fbk_nb_d(0)
1109:
1110:     Tnb_wall(0) = 1.0D-4
1111:     if(pr.ge.2) write(6,*) 'Tnb_wall=',Tnb_wall(0)
1112:
1113:     Tnb_wall_mob(0) = mu_wall_ * Tnb_wall(0)
1114:     if(pr.ge.2) write(6,*) 'Tnb_wall_mob=',Tnb_wall_mob(0)
1115:
1116:     Tnb_tot(0) = Tnb_wall(0) + Tnb_cryo_tot(0)
1117:     if(pr.ge.2) write(6,*) 'Tnb_tot=',Tnb_tot(0)
1118:
1119:     Tnb_tot_mob(0) = Tnb_wall_mob(0) + Tnb_cryo_mob(0)
1120:     if(pr.ge.2) write(6,*) 'Tnb_tot_mob=',Tnb_tot_mob(0)
1121:
1122:     Fplas_nb_(0) = ( kiss_mid_ * Fiss_mid_(0) + Fin_st_d(0) ) *
1123:    &      eta_nb_tot
1124:     if(pr.ge.2) write(6,*) 'Fplas_nb_=',Fplas_nb_(0)
1125:
1126:     Fplas_nb_t(0) = kiss_mid_ * Fiss_mid_(0) * Xiss_mid_t* eta_nb_tot
1127:     if(pr.ge.2) write(6,*) 'Fplas_nb_t=',Fplas_nb_t(0)
1128:
1129:     Fplas_nb_p(0) = Fplas_nb_(0) - Fplas_nb_t(0) - Fplas_nb_d(0)
1130:     if(pr.ge.2) write(6,*) 'Fplas_nb_p=',Fplas_nb_p(0)
1131:*
1132:* --- the following variables are set here ---
1133:*
1134:
1135:     Fuel_d(0) = Fplas_in_d(0) + Fgas_d(0) + Fplas_nb_d(0) +
1136:    &      Fplas_prop_d(0)
1137:     if(pr.ge.2) write(6,*) 'Fuel_d=',Fuel_d(0)
1138:
1139:     Fex_d(0) = Fuel_d(0) - Burn_d(0)
1140:     if(pr.ge.2) write(6,*) 'Fex_d=',Fex_d(0)
1141:
1142:     Fuel_p(0) = Fplas_in_p(0) + Xgas_p(0) * Fgas_tot(0) +
1143:    &      Fplas_prop_p(0)
1144:     if(pr.ge.2) write(6,*) 'Fuel_p=',Fuel_p(0)
1145:
1146:     Fex_p(0) = Reac_p(0) + Fuel_p(0)
1147:     if(pr.ge.2) write(6,*) 'Fex_p=',Fex_p(0)
1148:
1149:     Xnb_t(0) = Xnb_cryo_t(0)
1150:     if(pr.ge.2) write(6,*) 'Xnb_t=',Xnb_t(0)
1151:
1152:     Xnb_d(0) = Xnb_cryo_d(0)
1153:     if(pr.ge.2) write(6,*) 'Xnb_d=',Xnb_d(0)
1154:
1155:     Fsto_tri_in(0) = ( 1.0D0 - linj_bot_iss - mgas_bot_iss ) *
1156:    &      Fiss_bot_(0)
1157:     if(pr.ge.2) write(6,*) 'Fsto_tri_in=',Fsto_tri_in(0)
1158:
1159:     Fsto_deu_in(0) = ( 1.0D0 - kiss_mid_ - linj_mid_iss -
1160:    &      mgas_mid_iss ) * Fiss_mid_(0)
1161:     if(pr.ge.2) write(6,*) 'Fsto_deu_in=',Fsto_deu_in(0)
1162:
1163:     Fsto_pro_in(0) = ( 1.0D0 - linj_p ) * Fiss_top_(0)
1164:     if(pr.ge.2) write(6,*) 'Fsto_pro_in=',Fsto_pro_in(0)
1165:C ...
1166:C
1167:     jerr = 0
1168:
1169:C ... print out
1170:     write(6,*) '===initial values'
1171:     do i=1,nvar
1172:       write(6,711) 'X(0,',i,') <',varname(i),'> =',XVARS(0,i)
1173:     end do
1174:c
1175:c ... pump
1176:     do ip = 1, NT_PUMP
1177:C***       Tv_s(ip) =  0.0D0
1178:C***       Tp_reg_i(ip) = 0.0D0
1179:C***       Dp_reg_i(ip) = 0.0D0
1180:C***       Pp_reg_i(ip) = 0.0D0
1181:C***       Hep_reg_i(ip) = 0.0D0
1182:c
1183:       Tpv_i(0,ip)    = 1.0D-4 / NT_PUMP
1184:       Rt_rem_i(0,ip) = Tpv_i(0,ip) * emrRsAp
1185:       Dpv_i(0,ip)    = 1.0D-4 / NT_PUMP
1186:       Rd_rem_i(0,ip) = Dpv_i(0,ip) * emrRsAp
1187:       Ppv_i(0,ip)    = 1.0D-4 / NT_PUMP
1188:       Rp_rem_i(0,ip) = Ppv_i(0,ip) * emrRsAp
1189:       Hepv_i(0,ip)   = 1.0D-4 / NT_PUMP
1190:       Rhe_rem_i(0,ip) = Hepv_i(0,ip) * emrRsAp
1191:     end do
1192:
1193:c ... procession condition check
1194:     do i = 1,3
1195:       mcycle(i)   = 0
1196:       stime(i)    = 0.d0
1197:       scondvar(i) = 0.d0
1198:     end do
1199:c
1200:     tend = 0.d0
1201:     Timp_tend = Timp_(0)
1202:c
1203:  711 format(1x,a,i3,a,a,a,1p,d13.5)
1204:     return
1205:     end
```

● _cntinp.f

```
 1:     subroutine controlinput
 2:C=====================================================================
 3:C      Fusion reactor tritium inventory evaluation system
 4:C---------------------------------------------------------------------
 5:C  input problem control data
 6:C
 7:C  first version Feb 1997:
 8:C  update:
 9:C---------------------------------------------------------------------
10:C $Id: _cntinp.f,v 1.10 1998/03/26 05:22:22 sasaki Exp $
11:C=====================================================================
12:     implicit real*8 (A-H,O-Z)
13:C
14:     include '_sizes.h'
15:C
16:C    .... including common areas here and not passing by arguments
17:C        is only to use NAMELIST input for variables in them.
18:C
19:     include '_commons1.h'
20:C
21:     include 'varconst.h'
22:C
23:C----** local data **-------------------------------------------------
24:C
25:      integer nstartcycle, nstartpoint, nsuptime, nsdowntime
26:C
27:C*********************************************************************
28:C  << control data specification >>
29:C
30:C   Input data are specified by NAMELIST input.
31:C   Input namelist blocks in the following order.
32:C
33:C   For convinience to users, current input allows comment lines
34:C   starting with '#'.
35:C
36:C  ----------------------------
37:C  *  namelist $FILES  ...  $END   (FILENAMEINPUT routine input this)
38:C  ----------------------------
39:C
40:C    Specify filenames one per one line;
41:C
42:C      PRINT = 'printout-file-name'
43:C      RESULT = 'calculation result output directory/folder'
44:C      JDEBUG = <printout debugging information if non zero>
45:C      OUTPUTALL = <output results of all variables ignoring $OUTPUT>
46:C
47:C  ----------------------------
48:C  *  namelist $TIME  ...  $END    (this routine inputs)
49:C  ----------------------------
50:C
51:C    Simulation time & time step information.
52:C
53:C      TMAX = total-simulation-time  (in second)
54:C
55:C      NTIMES    = number of time intervals
56:C      TIMES()   = time interval boundaries (TIMES(1) must be 0)
57:C
58:C      NTIMEZONE = number of time zones
59:C      NSTEPS()  = number of time intervals in each time zone
60:C      TZBOUND() = boundaries of time zones.(NTIMEZONE+1 entries)
61:C
62:C      NCYCLE = cycle number
63:C
64:C      NBREAKC = break point cycle( max : MAXRES )
65:C      NBREAKP = break point sec  ( max : MAXRES )
66:C
67:C    * Users should input in either NTIMES-TIMES combination or
68:C      NTIMEZONE-NSTES-TZBOUND combination.
69:C
70:C   Time values are convertd into hour for calculation
71:C
72:C  ----------------------------------------------
73:C  *  $OUTPUT ... $END   (this routine inputs)
74:C  ----------------------------------------------
75:C
76:C    Specify variable names whose values of all time steps are output
77:C    as a comma separated list of names.
78:C
79:C      variable1, variable2, variable3, ..., variablen
80:C
81:C    Can be input any number of lines and a line having "$END" on
82:C    the head terminates this section..
83:C
84:C  ----------------------------------------------
85:C  *  $CONSTANT ... $END   (this routine inputs)
86:C  ----------------------------------------------
87:C
88:C    Change constant value if necessary.
89:C    Repeat lines as follows;
90:C
91:C      constant-name = value
92:C
93:C    Can be input none or any number of lines and a line having "$END" on
94:C    the head terminates this section..
95:C
96:C  ----------------------------------------------
97:C  *  $INITIAL ... $END   (INITIALINP routine inputs)
98:C  ----------------------------------------------
99:C
100:C    Change initial value if necessary.
101:C    Repeat lines as follows;
102:C
103:C      variable-name = value
104:C
105:C    Can be input none or any number of lines and a line having "$END" on
106:C    the head terminates this section..
107:C
108:C*********************************************************************
```

```
109:C
110:     namelist /TIME/ MODE,
111:    &            TMAX,NTIMES,TIMES,NTIMEZONE,NSTEPS,TZBOUND,
112:    &            NSUBSTEPS, NCYCLE, NBREAKC, NBREAKP,
113:    &            IABXCTL
114:CCC  namelist /OUTPUT/ NVAROUT, VAROUT
115:c    namelist /RESP/ nrespst, nresptime, deltas, eps
116:     namelist /RESP/ nstartcycle, nstartpoint, nsuptime, nsdowntime,
117:    &            deltas, eps
118:C ... name & value pairs to modify constants or initial value
119:C
120:     common /TEMPNAME/ NAME
121:     common /TEMPVAL/ VAL
122:     parameter (MAXTEMPNM = 64)
123:     character*16 NAME(MAXTEMPNM)
124:     double precision VAL(MAXTEMPNM)
125:C
126:     namelist /CONSTANT/ NAME, VAL
127:C
128:     character*80 LINE
129:C
130:C--------** input time specifications **--------------------------------
131:C
132:     MODE = 'RUN'
133:     NCYCLE = 79
134:     do i=1,MAXRES
135:       NBREAKC(i) = 0
136:       NBREAKP(i) = 0
137:       DBREAKP(i) = 0.0d0
138:     end do
139:     IABXCTL = 1
140:     NTIMEZONE = 0
141:     TIMES(0)  = 0.0d0
142:     TZBOUND(0)  = 0.0d0
143:     do i=1,MAXTZONE
144:       NSUBSTEPS(i) = 1
145:     end do
146:C
147:     write(6,*) ' ----------------------------------'
148:     write(6,*) ' ... input TIME specififcations ...'
149:     write(6,*) ' ----------------------------------'
150:C
151:     NTIMES = 0
152:
153:     read(50,nml=time)
154:     write(6,*) 'time read'
155:C
156:     if( mode(1:3) .eq. 'res' ) then
157:          mode = 'RES'
158:     else if( mode(1:3) .eq. 'run' ) then
159:          mode = 'RUN'
160:     end if
161:     if( mode(1:3) .ne. 'RUN' .and. mode(1:3) .ne. 'RES' )
162:    &                          mode = 'RUN'
163:
164:     nerr = 0
165:     write(6,*) 'MODE=', MODE
166:c
167:     if( MODE(1:3) .eq. 'RES' )    goto 1000
168:C-----------------------------------------------------------------
169:C            NORMAL RUN MODE
170:C-----------------------------------------------------------------
171:     nstartc = 1
172:C
173:C ... timezone is specified ................
174:C
175:     call settimes( nerr )
176:c
177:     do i=1,MAXRES
178:       if( nbreakc(i) .gt. 0 ) then
179:         dbreakp(i) = nbreakp(i)/3600.d0
180:         if( dbreakp(i) .gt. tmax ) then
181:           write(6,*) 'XXX NBREAKP is not valid.'
182:           nerr = nerr + 1
183:         else
184:           nbreakp(i) = ntimes
185:           do j=0,ntimes
186:             if( times(j) .gt. dbreakp(i) ) then
187:               nbreakp(i) = j
188:               goto 300
189:             end if
190:           end do
191: 300        continue
192:         end if
193:       end if
194:     end do
195:     write(6,*) 'START POINT---',nstartc,' cycle  '
196:     do i=1,MAXRES
197:       if( nbreakc(i) .ne. 0 ) then
198:         write(6,'(A,I5,A,I5,A,1PE12.5,A,I5,A)') ' BREAK POINT---',
199:    &        i,' ',nbreakc(i),' cycle   ',
200:    &        dbreakp(i)*3600d0,' sec   ', nbreakp(i),' step'
201:       end if
202:     end do
203:c
204:c ...   input file skip  &RESP cards
205:     read(50,nml=resp)
206:c
207:C
208:C--------** output variable specifications **-------------------------
209:C
210: 390  NVAROUT = 0
211:     do i=1,maxvarout
212:       varout(i) = ' '
213:     end do
214:C
215:     write(6,*) ' ----------------------------------------'
216:     write(6,*) ' ... input variable names to be output ...'
217:     write(6,*) ' ----------------------------------------'
218:     write(6,*) ' (number of total defined variables = ',nvar,')'
219:
220:check %%%%%%
221:c    do i=1,nvar
222:c    write(6,*) ' varname(',i,') <',varname(i),'>'
```

```
223:c    end do
224:c %%%%%%%%%%
225:C
226:cccc  read(50,nml=output)
227:c
228:     line=' '
229:     read(50,'(a)') line
230:     if( line(1:7).ne.'&OUTPUT' .and. line(1:7).ne.'$OUTPUT') then
231:       write(6,*) 'xxx &OUTPUT or $OUTPUT is not found.'
232:       write(6,*) 'xxx line: ',line
233:       stop 888
234:     end if
235:c
236:     nvarout = 0
237:c
238:  400 line = ' '
239:     read(50,'(a)',end=1900) line
240:     if( line(1:4).ne.'&END' .and. line(1:4).ne.'$END') then
241:       i1 = 1
242:       do i=1,len(line)
243:
244:         k = index(line(i1:),',')
245:         nvarout = nvarout + 1
246:
247:         if( nvarout.gt.maxvarout ) then
248:           write(6,*) 'xxx number of output variable NVAROUT = ',
249:    &        NVAROUT,
250:    &        'exceeds program limit (MAXVAROUT=',MAXVAROUT,')'
251:           stop 999
252:         end if
253:
254:         if( k.eq.0 ) then
255:           i2 = len(line)
256:         else
257:           i2 = i1+k-2
258:         end if
259:         if(line(i1:i2).eq.' ' ) then
260:           nvarout = nvarout - 1
261:         else
262:           varout(nvarout) = line(i1:i2)
263:         end if
264:         i1 = i2 + 2
265:         if( i1.gt.len(line) ) goto 410
266:       end do
267:  410   continue
268:       goto 400
269:     endif
270:C
271:C  ... remove unnecessary blanks
272:C
273:     do i=1,nvarout
274:       call CCOMP( VAROUT(i), len(varout(i)), varout(i), IFL )
275:     end do
276:c
277:c  ... match with VARNAME entry ...
278:c
279:     do 120 i=1,nvarout
280:c
281:       do j=1,i-1
282:         if( varout(j).eq.varout(i) ) then
283:           write(6,*) '!!! VAROUT ',varout(i),
284:    &          ' specified more than once.'
285:           ivarout(i) = 0
286:           goto 120
287:         end if
288:       end do
289:c
290:       do j=1,nvar
291:         kk = index(varname(j),'(t)')
292:         if( kk.gt.0 ) then
293:           kk = kk - 1
294:         else
295:           kk = len(varname(j))
296:         end if
297:         if( varname(j)(:kk).eq.varout(i) ) then
298:           ivarout(i) = j
299:           goto 130
300:         end if
301:       end do
302:c
303:       write(6,*) 'xxx ',varout(i),' is not in variable name list.'
304:       ivarout(i) = -1
305:       nerr = nerr + 1
306:       goto 120
307:c
308:  130   write(6,'(1x,a,i3,a,a,a,i4,a)')
309:    &     ' VAROUT(',i,') : ',varout(i),' (X(',ivarout(i),'))'
310:c
311:  120 continue
312:c
313:     kk = 0
314:
315:     do i=1,nvarout
316:       if( ivarout(i).gt.0 ) then
317:         kk = kk + 1
318:         if( kk.ne.i ) varout(kk) = varout(i)
319:       end if
320:     end do
321:
322:     nvarout = kk
323:     write(6,*) ' ... ',nvarout,' variables are output as result.'
324:C
325:     if(OUTPUTALL.ne.0) then
326:       write(6,*)
327:    &     '!!! all variables are output as calculation result.',
328:    &     ' Specifictions in $OUTPUT section will be igonored.'
329:     endif
330:C
331:C  ... test that files can be opened under specified "result" directory.
332:C
333:     iunit = 10
334:     call fileopen( 0, result(:llr), 'file000', iunit, ioerr )
335:     if( ioerr.ne.0 ) then
336:       write(6,*) 'XXX tried to open a file under specified output',
```

page 35

```
337:    &       ' directory, but failed.'
338:       write(6,*) '  Please check existence of the directory/folder ',
339:    &       'or access permission etc.'
340:c
341:       if( sysname.eq.'Macintosh') pause 'PAUSE'
342:c
343:       stop 888
344:     endif
345:c
346:     close (iunit, status='DELETE')
347:C
348:C--------** constant variable value re-definition **-----------------
349:C
350:     line=' '
351:     read(50,'(a)',end=1900) line
352:     if( line(1:9).ne.'&CONSTANT' .and. line(1:9).ne.'$CONSTANT') then
353:       write(6,*) 'xxx &CONSTANT or $CONSTANT is not found.'
354:       write(6,*) 'xxx line: ',line
355:       stop 888
356:     end if
357:c
358: 500 line = ' '
359:     read(50,'(a)',end=1900) line
360:     if( line(1:4).ne.'&END' .and. line(1:4).ne.'$END') then
361:
362:       k = index( line, '=' )
363:       if ( k.eq.0 ) then
364:         write(6,*) 'xxx constant change line must be "name = value"'
365:         write(6,*) 'xxx line: ',line
366:         stop 888
367:       end if
368:
369:       call changeconst( line(:k-1), line(k+1:), ierr )
370:
371:       goto 500
372:
373:     endif
374:C
375:C ... print out
376:C
377:     write(6,*) '=== constant values'
378:     do i=1,nconst
379:       write(6,711) 'C(',i,') <',constname(i),'> =',C(i)
380:     end do
381: 711 format(1x,a,i3,a,a,a,a,1p,d13.5)
382:C
383:C--------** initial value re-specification **-----------------
384:C
385:*    do i=1,MAXTEMPNM
386:*      NAME(i) = ' '
387:*      VAL(i) = 0.0d0
388:*    end do
389:*
390:*    read(50,nml=initial)
391:c
392:*    do i=1,MAXTEMPNM
393:*      if(NAME(i).ne.' ') then
394:*        do j=1,nvar
395:*          kk = index(varname(j),'(')
396:*          if( kk.eq.0 ) kk = len(varname(j))
397:*          if( varname(j)(:kk).eq.name(i) ) then
398:*            goto 300
399:*          end id
400:*        end do
401:*        write(*,*) 'XXX try to change initial value of <',name(i),
402:*   &       '> but no such a variable exists.'
403:*        nerr = nerr + 1
404:*        goto 320
405:C
406:* 300     write(*,*) '== initial value of <',name(i),
407:*   &       '> changed from ', X(j),' to ',val(i)
408:*        X(j) = val(i)
409:C
410:*      endif
411:*    enddo
412:C
413:* 320     continue
414:     goto 2000
415:c
416:c
417:C-----------------------------------------------------------------
418:C     RESPONSE MODE
419:C      TIME, result, OUTPUT, Variables  set by RESPONSE-FILE
420:C-----------------------------------------------------------------
421:c
422: 1000 continue
423:     write(6,*) 'response file read'
424:     kout = 20
425:     open(kout, file=response,  status='UNKNOWN',
426:    &         form='FORMATTED', iostat = ios )
427:     if( ios .ne. 0 )  then
428:       write(6,*) 'XXX failed to open response file '
429:       if( sysname.eq.'Macintosh' ) pause 'PAUSE'
430:       stop 888
431:     endif
432:c....... skip 2records
433:     read(kout,*)  line
434:     read(kout,*)  line
435:     read(kout,'(12X,E12.5)') tmax1
436:     if( tmax .eq. 0.d0 )   tmax = tmax1/3600.d0
437:c....... output variables
438:     read(kout,'(7X,A)')  result_o
439:     read(kout,'(20X,I5)')  nvarout
440:     write(6,*) 'nvarout=',nvarout
441:     if( nvarout .eq. 0 )  then
442:       ioutput = 1
443:     else
444:       ioutput = 0
445:       do i=1, nvarout
446:         read(kout,'(I5,3X,A32)') ivarout(i), varout(i)
447:c        write(6,*) '     ',ivarout(i), '  ', varout(i)
448:       end do
449:     endif
450:c....... constant values
```

```
451:      read(kout,'(20X,I5)')  nconst
452:      write(6,*) 'nconst=',nconst
453:      do i=1, nconst
454:        read(kout,'(A32,E15.8)') constname(i), C(i)
455:c        write(6,*) '      ',constname(i), C(i)
456:      end do
457:
458:c....... &RESP read
459:      nrespst(1) = 80
460:      nrespst(2) = 0
461:      nresptime(1) = 20
462:      nresptime(2) = 20
463:c     deltas = 2.0d+11
464:      deltas = 0.05
465:      eps    = 0.01
466:
467:      read(50,nml=resp)
468:      nrespst(1) = nstartcycle
469:      nrespst(2) = nstartpoint
470:      nresptime(1) = nsuptime
471:      nresptime(2) = nsdowntime
472:
473:      write(6,'(4(A,I5),2(A,1PE12.5))')
474:     &         'STEP RESPONSE   start=',NRESPST(1),' cycle   ',
475:     &          NRESPST(2),' sec   end 1 time',NRESPTIME(1),
476:     &         ' sec    end 2 time',NRESPTIME(2),
477:     &         ' sec   delta_S',DELTAS,' %      EPS',EPS
478:
479:      write(6,*) 'OUTPUTALL', OUTPUTALL,'  tmax',tmax,' hour'
480:      write(6,*) 'Result=',result
481:      write(6,*) 'Result old=',result_o
482:      write(6,*) 'IABXCTL=',iabxctl
483:      read(kout,'(20X,I5)')  NVAR
484:      write(6,*) 'NVAR=',NVAR
485:      nstartc = 0
486:      nstartsec = 0
487:      do i = 1, MAXRES
488:        read(kout,*, end=1300)  line
489:        read(kout,'(12X,I5)')  ic
490:        read(kout,'(12X,I5)')  ip
491:        read(kout,'(12X,E12.5)') tend
492:        read(kout,'(12X,E12.5)') Timp_tend
493:c........... variables
494:        nstartc = ic
495:        nstartsec = ip
496:        do j= 1, NVAR
497:          read(kout,'(A32,2E12.5)') VARNAME(j),XVARS(0,j),XVARS(1,j)
498:c         write(6,'(A32,2E12.5)') VARNAME(j),XVARS(0,j)XVARS(1,j)
499:        end do
500:c
501:        read(kout,'(A20)')  line
502:        read(kout,'(A20)')  line
503:        read(kout,'(24E12.5)') (Tpv_i(0,j),j=1,NT_PUMP)
504:c       write(6,'(A20,1P24E12.5)') line,(Tpv_i(0,j),j=1,NT_PUMP)
505:        read(kout,'(24E12.5)') (Tpv_i(1,j),j=1,NT_PUMP)
506:c       write(6,'(A20,1P24E12.5)') line,(Tpv_i(1,j),j=1,NT_PUMP)
507:        read(kout,'(A20)')  line
508:        read(kout,'(24E12.5)') (Dpv_i(0,j),j=1,NT_PUMP)
509:c       write(6,'(A20,1P24E12.5)') line,(Dpv_i(0,j),j=1,NT_PUMP)
510:        read(kout,'(24E12.5)') (Dpv_i(1,j),j=1,NT_PUMP)
511:c       write(6,'(A20,1P24E12.5)') line,(Dpv_i(1,j),j=1,NT_PUMP)
512:        read(kout,'(A20)')  line
513:        read(kout,'(24E12.5)') (Ppv_i(0,j),j=1,NT_PUMP)
514:c       write(6,'(A20,1P24E12.5)') line,(Ppv_i(0,j),j=1,NT_PUMP)
515:        read(kout,'(24E12.5)') (Ppv_i(1,j),j=1,NT_PUMP)
516:c       write(6,'(A20,1P24E12.5)') line,(Ppv_i(1,j),j=1,NT_PUMP)
517:        read(kout,'(A20)')  line
518:        read(kout,'(24E12.5)') (Hepv_i(0,j),j=1,NT_PUMP)
519:c       write(6,'(A20,1P24E12.5)') line,(Hepv_i(0,j),j=1,NT_PUMP)
520:        read(kout,'(24E12.5)') (Hepv_i(1,j),j=1,NT_PUMP)
521:c       write(6,'(A20,1P24E12.5)') line,(Hepv_i(1,j),j=1,NT_PUMP)
522:        read(kout,'(A20)')  line
523:        read(kout,'(24E12.5)') (Rt_rem_i(0,j),j=1,NT_PUMP)
524:c       write(6,'(A20,1P24E12.5)') line,(Rt_rem_i(0,j),j=1,NT_PUMP)
525:        read(kout,'(24E12.5)') (Rt_rem_i(1,j),j=1,NT_PUMP)
526:c       write(6,'(A20,1P24E12.5)') line,(Rt_rem_i(1,j),j=1,NT_PUMP)
527:        read(kout,'(A20)')  line
528:        read(kout,'(24E12.5)') (Rd_rem_i(0,j),j=1,NT_PUMP)
529:c       write(6,'(A20,1P24E12.5)') line,(Rd_rem_i(0,j),j=1,NT_PUMP)
530:        read(kout,'(24E12.5)') (Rd_rem_i(1,j),j=1,NT_PUMP)
531:c       write(6,'(A20,1P24E12.5)') line,(Rd_rem_i(1,j),j=1,NT_PUMP)
532:        read(kout,'(A20)')  line
533:        read(kout,'(24E12.5)') (Rp_rem_i(0,j),j=1,NT_PUMP)
534:c       write(6,'(A20,1P24E12.5)') line,(Rp_rem_i(0,j),j=1,NT_PUMP)
535:        read(kout,'(24E12.5)') (Rp_rem_i(1,j),j=1,NT_PUMP)
536:c       write(6,'(A20,1P24E12.5)') line,(Rp_rem_i(1,j),j=1,NT_PUMP)
537:        read(kout,'(A20)')  line
538:        read(kout,'(24E12.5)') (Rhe_rem_i(0,j),j=1,NT_PUMP)
539:c       write(6,'(A20,1P24E12.5)') line,(Rhe_rem_i(0,j),j=1,NT_PUMP)
540:        read(kout,'(24E12.5)') (Rhe_rem_i(1,j),j=1,NT_PUMP)
541:c       write(6,'(A20,1P24E12.5)') line,(Rhe_rem_i(1,j),j=1,NT_PUMP)
542:
543:c           NRESPST(1):cycle  NRESPST(2):sec
544:        if( NRESPST(1) .lt. ic ) goto 1300
545:        if( NRESPST(1) .eq. ic .and. NRESPST(2) .le. ip ) goto 1300
546:      end do
547: 1300   continue
548:      close(kout)
549:c
550:      if( (NRESPST(1) .ne. nstartc) .or.
551:     &   (NRESPST(2) .ne. nstartsec) )   then
552:        if( nstartc .eq. 0 )  then
553:          write(6,*) 'no data in file:',response
554:          stop 888
555:        else
556:          write(6,*) 'WARNING ***** start point not decided',
557:     &        ' use  ',nstartc,' cycle   ',nstartsec,' sec'
558:        end if
559:      else
560:        write(6,*) 'start point',nstartc,' cycle   ',
561:     &            nstartsec,' sec'
562:      end if
563:c....... break point clear
564:      do i = 1, MAXRES
```

```
565:       nbreakc(i) = 0
566:       dbreakp(i) = 0.d0
567:       nbreakp(i) = 0
568:     end do
569:     tend = tend / 3600d0
570:c
571:     call respinit()
572:c
573:     call settimes( nerr )
574:c
575:c ...      &OUTPUT  &CONSTANT  cards
576:     goto 390
577:c
578:c
579:C----** error stop **-------------------------------------------------
580:C
581: 2000 if( nerr.gt.0 ) then
582:       write(6,*)
583:    & 'XXX Fatal error detected in $TIME,$OUTPUT & $CONSTANT input.'
584:       if( sysname.eq.'Macintosh' ) then
585:         write(*,*)
586:    & 'XXX Fatal error detected in $TIME,$OUTPUT & $CONSTANT input.'
587:         pause
588:       end if
589:       stop 888
590:     end if
591:C
592:     return
593:C
594: 1900 write(6,*) 'xxx unexpected end of file on input.'
595:     stop 888
596:     end
597:C
598:C
599:     subroutine ccomp( outch, ln, inch, ifl )
600:C
601:C=================================================================
602:C PURPOSE:  'COMPRESS' CHARACTER STRING INCH BY REMOVING BLANK
603:C         INTO CHARACTER STRING 'OUTCH' UPTO LN CHARACTERS.
604:C         ('OUTCH' & 'INCH' CAN OVERLAP IN MEMORY BUT
605:C          THE STARTING ADDRESS OF 'OUTCH' MUST PRECEED OR BE SAME AS
606:C          THAT OF 'INCH'. )
607:C
608:C         IFL : FLAG
609:C           0 = NORMAL END
610:C           1 = NUMBER OF NON-BLANK CHARACTERS IN INCH EXEEDS LN.
611:C=================================================================
612:     character*(*)  outch, inch
613:     character*1    ch1
614:c
615:     ifl  = 0
616:     ll   = len(inch)
617:     nn   = 0
618:     do 100 i=1,ll
619:       if( inch(i:i).ne.' ' ) then
620:         nn  = nn + 1
621:         if( nn.le.ln ) then
622:           ch1 = inch(i:i)
623:           outch(nn:nn) = ch1
624:         endif
625:       endif
626: 100 continue
627:     if( nn.lt.ln ) outch(nn+1:ln) = ' '
628:     if( nn.gt.ln ) ifl = 1
629:     return
630:     end
631:
632:
633:
634:
635:     subroutine settimes( nerr )
636:C
637:C=================================================================
638:C PURPOSE:  READ INPUT-FILE
639:C         SET  TIME-STEPS
640:C=================================================================
641:     implicit real*8 (A-H,O-Z)
642:C
643:     include '_sizes.h'
644:C
645:     include '_commons1.h'
646:C
647:     include 'varconst.h'
648:
649:
650:     if( NTIMEZONE.ne.0 ) then
651:C
652:       if( ntimezone.gt.maxtzone ) then
653:         write(6,*) 'xxx number of time zones NTIMEZONE = ',
654:    &           ntimezone,
655:    &      'exceeds program limit (MAXTZONE=',MAXTZONE,')'
656:         stop 999
657:       end if
658:C
659:       write(6,*) 'NTIMEZONE = ',NTIMEZONE
660:C
661:       do i=1,ntimezone
662:C
663:         write(6,7100) i,NSTEPS(I),
664:    &        'tzbound(',i-1,') tzbound(',i,') : ',
665:    &                tzbound(i-1),tzbound(i)
666:         write(6,7102) NSUBSTEPS(I)
667:C
668: 7100     format(1x,'Timezone ',i3,' : ',i4,' time steps'
669:    &         /5x,a,i3,a,i3,a,1p,2d12.5)
670: 7102     format(1x,'   each step is divided into ',i5,' substeps.')
671:C
672:         if( nsteps(i).le.0 ) then
673:           write(6,*) 'xxx NSTEPS(',i,') must be positive but ',
674:    &          nsteps(i)
675:           nerr = nerr + 1
676:         endif
677:         if( nsubsteps(i).le.0 ) then
678:           write(6,*) 'xxx NSUBSTEPS(',i,') must be positive but ',
```

```
679:    &           nsubsteps(i)
680:          nerr = nerr + 1
681:        endif
682:C
683:        if( tzbound(i).lt.0.0d0 .or. tzbound(i).le.tzbound(i-1) )
684:    &           then
685:          write(6,*) 'xxx invalid time zone boundary (zone ',i,')'
686:          write(6,*) ' tzbound(',i,') tzbound(',i-1,') : ',
687:    &                tzbound(i),tzbound(i-1)
688:          nerr = nerr + 1
689:        end if
690:C
691:        if( nsteps(i).ne.0 ) then
692:          ddt = (tzbound(i)-tzbound(i-1))/nsteps(i)
693:          write(6,*) '     step width ',ddt,' (sec) ',ddt/3600.0,
694:    &             ' (hour)'
695:        end if
696:C
697:        k = ntimes
698:        if( k.lt.maxtimes ) then
699:          times(k) = tzbound(i-1)
700:          do j=1,nsteps(i)
701:            if( k+j.lt.maxtimes ) then
702:              times(k+j) = times(k)  + j*ddt
703:            end if
704:          end do
705:        end if
706:        ntimes = ntimes + nsteps(i)
707:      end do
708:C
709:      if( ntimes.gt.maxtimes ) then
710:        write(6,*) 'xxx number of time steps NTIMES = ', ntimes,
711:    &     'exceeds program limit (MAXTIMES=',MAXTIMES,')'
712:        stop 999
713:      end if
714:
715:      write(6,*) 'NCYCLE = ', ncycle
716:      write(6,*) 'IABXCTL = ', iabxctl
717:      write(6,*) 'ntimes', ntimes, ' times', times(0),'   ',
718:    &            times(1),'   ',times(ntimes)
719:C
720:C ... no time zone specified .....
721:C
722:    else
723:
724:      if( NTIMES.le.0 ) then
725:        write(6,*) 'xxx number of time steps NTIMES = ', ntimes,
726:    &     'must be positive.'
727:        stop 999
728:      endif
729:
730:      if( ntimes.gt.maxtimes ) then
731:        write(6,*) 'xxx number of time steps NTIMES = ', ntimes,
732:    &     'exceeds program limit (MAXTIMES=',MAXTIMES,')'
733:        stop 999
734:      endif
735:
736:      do i=1,ntimes
737:        if( times(i).lt.0.0d0 .or. times(i).le.times(i-1) ) then
738:          write(6,*) 'xxx invalid time step boundary (step ',i,')'
739:          write(6,*) ' times(',i,') times(',i-1,') : ',
740:    &                times(i),times(i-1)
741:          nerr = nerr + 1
742:        end if
743:      end do
744:    end if
745:c
746:check %%%%%%
747:c    do i=0,ntimes
748:c    write(6,*) ' times(',i,') <',times(i),'>'
749:c    end do
750:c %%%%%%%%%%
751:C
752:    if( times(ntimes).ne.tmax ) then
753:      write(6,*) '!!! TMAX is not equal to the last time boundary'
754:      write(6,*) '   set TMAX to TIMES(NTIMES).'
755:      TMAX  =  TIMES(NTIMES)
756:    end if
757:c
758:c   ... convert times into hour.
759:c
760:    do i=0,ntimes
761:      times(i) = times(i)/3600.0d0
762:    end do
763:    tmax = tmax/3600.d0
764:C
765:    if( nerr.gt.0 ) then
766:      write(6,*) 'XXX time step information is not valid.'
767:    end if
768:
769:    return
770:    end
```

● _initinp.f

```
1:     subroutine initialinp
 2:C=====================================================================
 3:C      Fusion reactor tritium inventory evaluation system
 4:C---------------------------------------------------------------------
 5:C  input problem control data (2) : iniitial value change if any.
 6:C
 7:C  first version Feb 1997:
 8:C  update:
 9:C---------------------------------------------------------------------
10:C $Id: _initinp.f,v 1.2 1998/03/26 05:22:22 sasaki Exp $
11:C=====================================================================
12:     implicit real*8 (A-H,O-Z)
```

```
13:C
14:     include '_sizes.h'
15:C
16:C    .... including common areas here and not passing by arguments
17:C        is only to use NAMELIST input for variables in them.
18:C
19:     include '_commons1.h'
20:C
21:     include 'varconst.h'
22:C
23:C----** local data **-------------------------------------------------
24:C
25:     character*80 LINE
26:C
27:C--------** constant variable value re-definition **-----------------
28:C
29:c
30:     line=' '
31:     read(50,'(a)',end=1900) line
32:     if( line(1:8).ne.'&INITIAL' .and. line(1:8).ne.'$INITIAL') then
33:       write(6,*) 'xxx line is not &INITIAL or $INITIAL.'
34:       write(6,*) 'xxx line: ',line
35:       if( sysname.eq.'Macintosh' ) then
36:         write(*,*) 'xxx line is not &INITIAL or $INITIAL.'
37:         write(*,*) 'xxx line: ',line
38:         pause
39:       end if
40:       stop 888
41:     end if
42:c
43:     nerr = 0
44:c
45:  500 line = ' '
46:     read(50,'(a)',end=1900) line
47:     if( line(1:4).ne.'&END' .and. line(1:4).ne.'$END') then
48:
49:       k = index( line, '=' )
50:       if ( k.eq.0 ) then
51:         write(6,*) 'xxx line must be "name = value"'
52:         write(6,*) 'xxx line: ',line
53:         if( sysname.eq.'Macintosh' ) then
54:           write(*,*) 'xxx line must be "name = value"'
55:           write(*,*) 'xxx line: ',line
56:           pause
57:         end if
58:         stop 888
59:       end if
60:
61:       call changeinitial( line(:k-1), line(k+1:), ierr )
62:       if (ierr.ne.0) nerr = nerr + 1
63:
64:       goto 500
65:
66:     endif
67:c
68:     if( nerr.gt.0 ) then
69:       write(6,*) 'xxx Fatal error detected in $INITIAL input.'
70:       if( sysname.eq.'Macintosh' ) then
71:         write(*,*) 'xxx Fatal error detected in $INITIAL input.'
72:         pause
73:       end if
74:       stop 888
75:     end if
76:C
77:
78:C
79: 1900 write(6,*) 'No initial value specification by user.'
80:
81:c
82:     return
83:     end
```

● _changes.f

```
 1:     subroutine changeconst( name, value, ierr )
 2:c=====================================================================
 3:c      Fusion reactor tritium inventory evaluation system
 4:c---------------------------------------------------------------------
 5:c    change constant value
 6:c---------------------------------------------------------------------
 7:c  $Id: _changes.f,v 1.2 1997/05/07 02:20:57 sasaki Exp $
 8:c=====================================================================
 9:     character*(*) name, value
10:c
11:     include '_sizes.h'
12:     include 'varconst.h'
13:c
14:     character*32 cwork
15:     double precision C0
16:c ---------------------------------------------------
17:c
18:     ierr = 0
19:c
20:c ... remove unnecessary blanks if any
21:c
22:     call ccomp( name, len(name), name, ifl )
23:     ln = index(name,' ')-1
24:     if( ln.lt.0 ) ln = len(name)
25:
26:     call ccomp( value, len(value), value, ifl )
27:     lv = index(value,' ')-1
28:     if( lv.lt.0 ) lv = len(value)
29:c
30:c ... search constants name list
31:c
32:     do in=1,nconst
33:       if( constname(in).eq.name(:ln) ) then
34:         goto 120
```

```
35:        endif
36:      end do
37:      write(6,*) 'xxx cannot find constant name <',name(:ln),'>'
38:      ierr = 1
39:      return
40:c
41:  120 continue
42:c
43:      if( lv.gt.len(cwork) ) then
44:        write(6,*) 'xxx value character string is too long <',
45:     &           value(:lv),'>'
46:        ierr = 1
47:        return
48:      endif
49:c
50:c ... convert value string into numerical value
51:c
52:      C0 = C(in)
53:      cwork = value(:lv)
54:      read( cwork, '(d32.0)', err=190 ) C(in)
55:c
56:      write(6,*) '== value of constant <',name(:ln),'> is changed to ',
57:     & C(in),' from ',C0
58:      return
59:c
60:  190 write(6,*) 'xxx value form is invalid in <',name(:ln),' = ',
61:     &  value(:lv),'>'
62:      ierr = 1
63:      return
64:      end
65:c
66:c=====================================================================
67:c
68:      subroutine changeinitial( name , value, ierr )
69:c -----------------------------------------------------
70:c  change initial value
71:c -----------------------------------------------------
72:      character*(*) name, value
73:c
74:      include '_sizes.h'
75:      include 'varconst.h'
76:c
77:      character*32 cwork
78:      double precision C0
79:c -----------------------------------------------------
80:c
81:      ierr = 0
82:c
83:c ... remove unnecessary blanks if any
84:c
85:      call ccomp( name, len(name), name, ifl )
86:      ln = index(name,' ')-1
87:      if( ln.lt.0 ) ln = len(name)
88:
89:      call ccomp( value, len(value), value, ifl )
90:      lv = index(value,' ')-1
91:      if( lv.lt.0 ) lv = len(value)
92:c
93:c ... search variable name list
94:c
95:      do in=1,nvar
96:        kk = index(varname(in),'(t)')
97:        if( kk.gt.0 ) then
98:           kk = kk - 1
99:        else
100:          kk = len(varname(in))
101:       endif
102:       if( varname(in)(:kk).eq.name(:ln) ) then
103:         goto 120
104:       endif
105:     end do
106:     write(6,*) 'xxx cannot find variable name <',name(:ln),'>'
107:     ierr = 1
108:     return
109:c
110:  120 continue
111:c
112:     if( lv.gt.len(cwork) ) then
113:       write(6,*) 'xxx value character string is too long <',
114:    &          value(:lv),'>'
115:       ierr = 1
116:       return
117:     endif
118:c
119:c ... convert value string into numeric value
120:c
121:     C0 = XVARS(0,in)
122:     cwork = value(:lv)
123:     read( cwork, '(d32.0)', err=190 ) XVARS(0,in)
124:c
125:     XVARS(1,in) = XVARS(0,in)
126:c
127:     write(6,*) '== initial value of <',name(:ln),'> is changed to ',
128:    & XVARS(0,in),' from ',C0
129:     return
130:c
131:  190 write(6,*) 'xxx value form is invalid in <',name(:ln),' = ',
132:    &  value(:lv),'>'
133:     ierr = 1
134:c
135:     return
136:     end
```

● _solver.f

```
 1:     subroutine solver( tmax, ntimes, times, nsteps,
 2:    &  ntimezone, delta_t, method,
 3:    &  sysname, jdebug, jdisp,
 4:    &  F, ncycle, nstartc, nbreakc, nbreakp,
 5:    &  dnsgmT, mode, ntimes_r, times_r )
 6:c=========================================================================
 7:c      Fusion reactor tritium inventory evaluation system
 8:c-------------------------------------------------------------------------
 9:c $Id: _solver.f,v 1.7 1997/05/07 07:28:19 sasaki Exp $
10:c=========================================================================
11:c
12:     implicit real*8 (a-h,o-z)
13:c
14:     include '_sizes.h'
15:     include 'varconst.h'
16:c
17:     real*8 times(0:ntimes), times_r(0:ntimes)
18:     integer nsteps(ntimezone), ic
19:     integer nbreakc(*), nbreakp(*)
20:c
21:     character*(*) METHOD
22:     character*(*) SYSNAME
23:     character*(*) mode
24:
25:c***  character*24 the_date
26:c
27:     real*8 F(ndfvar)
28:c
29:c-------------------------------------------------------------------------
30:c ** set initial condition **
31:c-------------------------------------------------------------------------
32:c
33:     write(6,'(/1x,a)') '--- set initial condition:'
34:c
35:     delta_t = times(1) - times(0)
36:c
37:     lpr = 0
38:     if(jdebug.ne.0) lpr = 2
39:     if( mode .eq. 'RUN' ) then
40:       call VARINIT(delta_t, F, lpr, ierr )
41:       write(6,'(/1x,a)')
42:    &       '--- checking user specified initial condition:'
43:     end if
44:c
45:     if( mode .eq. 'RUN' ) then
46:       call INITIALINP
47:       write(6,'(/1x,a/)') '--- initial condition set complete:'
48:     end if
49:c
50:c    ... Initial_step must be 1 just for the first call not to
51:c       overwrite initial value settings ...
52:c
53:     Initial_step = 1
54:c
55:     J = 0
56:     dt1 = delta_t
57:c
58:c
59:     if( mode .eq. 'RES' )  goto 3000
60:c
61:C*****************************************************************
62:C*   mode = RUN
63:C*****************************************************************
64:c-----------------------------------------------------------------
65:c  ** NCYCLE loop **
66:c-----------------------------------------------------------------
67:     call rstoutfst( )
68:
69:     do ic = nstartc, ncycle
70:
71:       Initial_step = 1
72:c
73:c-----------------------------------------------------------------
74:c     ** timestep loop **
75:c-----------------------------------------------------------------
76:c
77:       do it = 1, ntimes
78:c
79:         t = (ic-1)*tmax+times(it)
80:         delta_t = times(it) - times(it-1)
81:
82:         if( it .eq. 1 ) then
83:c***        call fdate(the_date)
84:c***        write(6,*) 'CYCLE',ic,'  TIME=',the_date
85:           write(6,*) 'CYCLE',ic
86:           if( sysname.eq.'Macintosh') then
87:c***              write(*,*) 'CYCLE',ic,'  TIME=',the_date
88:             write(*,*) 'CYCLE',ic
89:           endif
90:         endif
91:         if( jdisp .ne. 0 ) then
92:           write(6,7000) it,times(it),times(it)*3600d0,
93:    &            t,t*3600d0,delta_t*3600d0
94: 7000 format(/1x,'   step',i4,' t=',1p,d12.5,'(',d11.4,
95:    & 'sec)---',1p,d12.5,'(',d11.4,' sec) dt = ',d10.3,' sec')
96:c
97:c  ... write(*,...) is on MRWE window for AbSoft FORTRAN77 for MAC
98:c
99:           if(sysname.eq.'Macintosh') then
100:            write(*,7000) it,times(it),times(it)*3600d0,
101:   &                t,t*3600d0,delta_t*3600d0
102:          endif
103:        endif
104:c=========================================================================
105:        call fdiffvect( It, J, t, delta_t, dt1, F,
106:   &              Initial_step, ntimes, ic, tmax,
107:   &              dnsgmT, mode )
108:c=========================================================================
109:
110:        Initial_step = 0
111:c
```

page 42

```
112:c      ... end of sub-timestep
113:          dt1 = delta_t
114:c
115:c  --- for debugging ----------------------------
116:c
117:          if( jdebug.ge.1 ) then
118:            do i=1,ndfvar
119:              ii = idfvar(i)
120:              if(F(I).ne.0.0d0) then
121:                TT = abs(XVARS(it-1,ii)/F(i))
122:              else
123:                TT = 1000.0
124:              endif
125:c
126:              write(6,7111)
127:    &             it,'% ',varname(ii)(1:16),' X(',ii,') : ',
128:    &             XVARS(it,ii), F(i), TT
129: 7111 format(1x,i4,a,a,a,i3,a,d11.4,' F: ',d11.4,' TT: ',d11.4)
130:c
131:            end do
132:            write(6,*) '%%%% refered variables from dX/dt %%%%'
133:            do i=1,nvar
134:              if( ivarflag(i).eq.2 )
135:    &           write(6,*) it,'% ',varname(i)(1:16),
136:    &                ' X(',i,') : ',XVARS(it,i)
137:            end do
138:            write(6,*) '%%%% other variables %%%%'
139:            do i=1,nvar
140:              if( ivarflag(i).eq.3.or.ivarflag(i).eq.4 )
141:    &           write(6,*) it,'% ',varname(i)(1:16),
142:    &                ' X(',i,') : ',XVARS(it,i)
143:            end do
144:          end if
145:c
146:C     ........... break point check
147:          do i = 1, MAXRES
148:            if (ic .eq. nbreakc(i) .and. it .eq. nbreakp(i)) then
149:c     ......... RESPONSE FILE write
150:              call rstoutput( i )
151:            end if
152:          end do
153:c
154:c      ... end of timestep
155:        end do
156:c
157:c
158:c
159:c  ==============================================================
160:c        output results
161:c  ==============================================================
162:c
163:c***    call fdate(the_date)
164:c***     write(6,*) '          ',
165:c*** &        'end-of-calculation   TIME=',the_date
166:c***     if( sysname.eq.'Macintosh' ) then
167:c***       write(*,*) '          ',
168:c*** &        'end-of-calculation   TIME=',the_date
169:c***     endif
170:        call OUTPUT( ic, 1 )
171:c
172:c  ==============================================================
173:c        variable reset
174:c  ==============================================================
175:c
176:        do ivar = 1, NVAR
177:          XVARS(0,ivar) = XVARS(ntimes,ivar)
178:        end do
179:        do ip = 1, NT_PUMP
180:          Tpv_i(0,ip)    = Tpv_i(ntimes,ip)
181:          Rt_rem_i(0,ip)  = Rt_rem_i(ntimes,ip)
182:          Dpv_i(0,ip)    = Dpv_i(ntimes,ip)
183:          Rd_rem_i(0,ip)  = Rd_rem_i(ntimes,ip)
184:          Ppv_i(0,ip)    = Ppv_i(ntimes,ip)
185:          Rp_rem_i(0,ip)  = Rp_rem_i(ntimes,ip)
186:          Hepv_i(0,ip)   = Hepv_i(ntimes,ip)
187:          Rhe_rem_i(0,ip) = Rhe_rem_i(ntimes,ip)
188:        end do
189:c
190:      end do
191:c
192:      goto 9000
193:c
194:c
195:C*****************************************************************
196:C*   mode = RES
197:C*****************************************************************
198: 3000 continue
199:c-----------------------------------------------------------------
200:c        S1 S0  step resp calculation
201:c-----------------------------------------------------------------
202:      ic = nstartc
203:      Initial_step = 0
204:c
205:      do is = 1, 2
206:c
207:        do it = 2, ntimes_r
208:c
209:          t = (ic-1)*tmax+times_r(it)
210:          delta_t = times_r(it) - times_r(it-1)
211:
212:          if( jdisp .ne. 0 ) then
213:            write(6,7000) it,times_r(it),times_r(it)*3600d0,
214:    &          t,t*3600d0,delta_t*3600d0
215:            if(sysname.eq.'Macintosh') then
216:              write(*,7000) it,times(it),times(it)*3600d0,
217:    &                t,t*3600d0,delta_t*3600d0
218:            endif
219:          endif
220:          call fdiffvect( It, J, t, delta_t, dt1, F,
221:    &              Initial_step, ntimes_r, ic, tmax,
222:    &              dnsgmT, mode )
223:
224:c      ... end of sub-timestep
225:          dt1 = delta_t
```

page 43

```
226:c
227:c      ... end of timestep
228:       end do
229:c
230:c ...... output results
231:c
232:       call OUTPUT( ic, 1 )
233:c
234:c ...... variable reset
235:c
236:       do ivar = 1, NVAR
237:         XVARS(1,ivar) = XVARS(ntimes_r,ivar)
238:       end do
239:       do ip = 1, NT_PUMP
240:         Tpv_i(1,ip)    = Tpv_i(ntimes_r,ip)
241:         Rt_rem_i(1,ip)  = Rt_rem_i(ntimes_r,ip)
242:         Dpv_i(1,ip)    = Dpv_i(ntimes_r,ip)
243:         Rd_rem_i(1,ip)  = Rd_rem_i(ntimes_r,ip)
244:         Ppv_i(1,ip)    = Ppv_i(ntimes_r,ip)
245:         Rp_rem_i(1,ip)  = Rp_rem_i(ntimes_r,ip)
246:         Hepv_i(1,ip)   = Hepv_i(ntimes_r,ip)
247:         Rhe_rem_i(1,ip) = Rhe_rem_i(ntimes_r,ip)
248:       end do
249:c
250:       if( is .eq. 1 )  call respdown()
251:c
252:     end do
253:c***  call fdate(the_date)
254:c***  write(6,*) 'Step-Responce calculation end  TIME=',
255:c*** &          the_date
256:c***  if(sysname.eq.'Macintosh') then
257:c***     write(*,*) 'Step-Responce calculation end  TIME=',
258:c*** &          the_date
259:c***  endif
260:c
261:c-----------------------------------------------------------------------
262:c        STATIC calculation
263:c-----------------------------------------------------------------------
264:c
265:     nstartp = 2
266:     call respreset( nstartp )
267:c
268:     do ic = nstartc, ncycle
269:
270:       Initial_step = 1
271:c
272:       do it = 1, ntimes
273:c
274:         if( it .lt. nstartp )   goto 3100
275:
276:         t = (ic-1)*tmax+times(it)
277:         delta_t = times(it) - times(it-1)
278:
279:         if( it .eq. 1 ) then
280:c***        call fdate(the_date)
281:c***        write(6,*) 'CYCLE',ic,'  TIME=',the_date
282:           write(6,*) 'CYCLE',ic
283:           if( sysname.eq.'Macintosh') then
284:c***          write(*,*) 'CYCLE',ic,'  TIME=',the_date
285:                write(*,*) 'CYCLE',ic
286:           endif
287:         endif
288:         if( jdisp .ne. 0 ) then
289:          write(6,7000) it,times(it),times(it)*3600d0,
290:     &              t,t*3600d0,delta_t*3600d0
291:c
292:c ... write(*,...) is on MRWE window for AbSoft FORTRAN77 for MAC
293:c
294:           if(sysname.eq.'Macintosh') then
295:             write(*,7000) it,times(it),times(it)*3600d0,
296:     &               t,t*3600d0,delta_t*3600d0
297:           endif
298:         endif
299:
300:         call fdiffvect( It, J, t, delta_t, dt1, F,
301:     &              Initial_step, ntimes, ic, tmax,
302:     &              dnsgmT, mode )
303:
304:c      ... end of sub-timestep
305:         dt1 = delta_t
306:c
307: 3100     continue
308:         Initial_step = 0
309:c
310:c      ... end of timestep
311:       end do
312:c
313:c ...... output results
314:c
315:c***    call fdate(the_date)
316:c***    write(6,*) '         end-of-calculation  TIME=',
317:c*** &          the_date
318:c***    if( sysname.eq.'Macintosh' ) then
319:c***      write(*,*) '         end-of-calculation  TIME=',
320:c*** &          the_date
321:c***    endif
322:       call OUTPUT( ic, nstartp )
323:c
324:c ...... variable reset
325:c
326:       do ivar = 1, NVAR
327:         XVARS(0,ivar) = XVARS(ntimes,ivar)
328:       end do
329:       do ip = 1, NT_PUMP
330:         Tpv_i(0,ip)    = Tpv_i(ntimes,ip)
331:         Rt_rem_i(0,ip)  = Rt_rem_i(ntimes,ip)
332:         Dpv_i(0,ip)    = Dpv_i(ntimes,ip)
333:         Rd_rem_i(0,ip)  = Rd_rem_i(ntimes,ip)
334:         Ppv_i(0,ip)    = Ppv_i(ntimes,ip)
335:         Rp_rem_i(0,ip)  = Rp_rem_i(ntimes,ip)
336:         Hepv_i(0,ip)   = Hepv_i(ntimes,ip)
337:         Rhe_rem_i(0,ip) = Rhe_rem_i(ntimes,ip)
338:       end do
339:c
```

```
340:        nstartp = 1
341:c
342:      end do
343:c
344:C*****************************************************************
345:C*   finish
346:C*****************************************************************
347: 9000 write(6,*) '=== calculation finished ==='
348:
349:      write(66,*) 'Max condition value'
350:      write(66,'(A,I3,A,1PE12.5,A,E15.8)')
351:     &   'Fiss_top_ Fiss_mid_ Fiss_bot_ CYCLE:',mcycle(1),
352:     &   ' time:',stime(1)*3600d0,' sec  value',scondvar(1)
353:      write(66,'(A,I3,A,1PE12.5,A,E15.8)')
354:     &   'Fsto_out_t Fsto_out_d       CYCLE:',mcycle(2),
355:     &   ' time:',stime(2)*3600d0,' sec  value',scondvar(2)
356:      write(66,'(A,I3,A,1PE12.5,A,E15.8)')
357:     &   'Fgas_in_tri Fgas_in_deu     CYCLE:',mcycle(3),
358:     &   ' time:',stime(3)*3600d0,' sec  value',scondvar(3)
359:c
360:      return
361:      end
```

● fdiff.f

```
 1:      subroutine fdiffvect( i, j, t, delta_t, dt1, F,
 2:     &                Initial_step, ntimes, ic, tmax,
 3:     &          dnsgmT, mode )
 4:C=========================================================
 5:C
 6:C Calculate dX/dt and other variables at time t.
 7:C.........................................................
 8:C  Time "t" is time(i) and variable(i-1) is
 9:C  at time(i-1) = time t - delta_t.
10:C.........................................................
11:C
12:C  i  : current time step no. (from 1).
13:C  j  : time step index of the last "tend(j)"
14:C  t  : current time.
15:C  delta_t  : time width of the current time step.
16:C  dt1  : time width of the last time step (if available).
17:C  F  : time differetiation coeeficients.
18:C  Initial_step  : set non-zero only for the first call
19:C
20:C*********************************************************
21:C  This file is created automatically.
22:C    Creation date : << Tue May 6 14:21:39 1997 >>
23:C*********************************************************
24:C
25:C=========================================================
26:
27:      implicit double precision (a-h,o-z)
28:C ... variables & constants definition ...
29:C
30:
31:      include '_sizes.h'
32:      include 'varconst.h'
33:
34:C ....... arguments ......
35:      double precision t,delta_t,tmax
36:      double precision F(ndfvar)
37:      integer ntimes, ic
38:      character*(*) mode
39:C........ functions ......
40:      double precision ai,rho,Rc,Ed,fm,sigma_tend
41:      double precision fnbpower
42:      double precision condvar
43:
44:      double precision condn
45:c........ work
46:      double precision A(3,3)
47:C---------------------------------------------------------
48:C
49:C   statement function
50:      ABX( aa,bb,xx ) = aa + bb * xx
51:C=========================================================
52:c
53:      condn = 1000d0
54:
55:      tcycle = mod ( t , tcycle_p )
56:      if( tcycle .eq. 0.d0 )  tcycle = t-(ic-1)*tcycle_p
57:
58:c     tpoint_cycle = t - Sigma_tend ( t , tend )
59:*
60:*#########################################################
61:*##(The Plasma)
62:*#########################################################
63:*
64:      if( mode .eq. 'RUN'  .or. mode .eq. 'RRR' )  then
65:        fpp = fppower ( tcycle , t_burn , t_p_ru , t_p_rd
66:     &                , t_dwell )
67:        if( t .lt. t_p_ru .and. fpp .lt. 0.1d0 ) fpp = 0.1d0
68:        Pfusion(i) = 1000D0 * fpp
69:        Burn_t(i) = 2.12D-3 * Pfusion(i)
70:      else
71:        Burn_t(i) = res_N(i) * dnsgmT
72:        Pfusion(i) = Burn_t(i) / 2.12D-3
73:        fpp =  Pfusion(i) / 1000D0
74:      end if
75:
76:      Fuel_t(i) = Burn_t(i) / fbu
77:
78:      RcM = Rc ( Medge , Tedge ) * fpp
79:
80:      ARcM = 1000D0 / 8760 * RcM / ( 0.5 * 3.016D0 )
81:
```

page 45

```
 82:     F(1) = ABX ( d_C_t(i-1) , 0.0d0 , C_t(i-1) )
 83:     call ABXdelta( d_C_t(i-1) , 0.0d0  , C_t, i, delta_t, 'C_t',
 84:    &            iabxctl)
 85:
 86:     F(2) = ABX ( d_C_d(i-1) , 0.0d0 , C_d(i-1) )
 87:     call ABXdelta( d_C_d(i-1) , 0.0d0  , C_d, i, delta_t, 'C_d',
 88:    &            iabxctl)
 89:
 90:     F(3) = ABX ( d_C_p(i-1) , 0.0d0 , C_p(i-1) )
 91:     call ABXdelta( d_C_p(i-1) , 0.0d0  , C_p, i, delta_t, 'C_p',
 92:    &           iabxctl)
 93:
 94:     EdM = Ed ( Medge , Tedge ) * fpp
 95:
 96:     EdMAd = EdM * Ad * dc * rho ( Medge ) / ( 0.5 * 3.016D0 * 8760 )
 97:
 98:     F(4) = ABX ( d_D_t(i-1) , 0.0d0 , D_t(i-1) )
 99:     call ABXdelta( d_D_t(i-1) , 0.0d0  , D_t, i, delta_t, 'D_t',
100:    &            iabxctl)
101:
102:     F(5) = ABX ( d_D_d(i-1) , 0.0d0 , D_d(i-1) )
103:     call ABXdelta( d_D_d(i-1) , 0.0d0  , D_d, i, delta_t, 'D_d',
104:    &             iabxctl)
105:
106:     F(6) = ABX ( d_D_p(i-1) , 0.0d0 , D_p(i-1) )
107:     call ABXdelta( d_D_p(i-1) , 0.0d0  , D_p, i, delta_t, 'D_p',
108:    &             iabxctl)
109:
110:     Fex_t(i) = Fuel_t(i) * ( 1 - fbu ) - ( d_Adiv_t(i-1) +
111:    &      d_Afw_t(i-1) ) - d_C_t(i-1) - d_D_t(i-1)
112:     if( Fex_t(i) .lt. 0.0d0 ) call negfix( Fex_t(i), 'Fex_t(i)',
113:    &      0.0d0 )
114:
115:     Fplas_in_t(i) = Fuelf_t(i-1) * fin_pellet
116:     if( Fplas_in_t(i) .lt. 0.0d0 ) call negfix( Fplas_in_t(i),
117:    &      'Fplas_in_t(i)', 0.0d0 )
118:
119:     Fgas_t(i) = Fuelf_t(i-1) * fin_gas
120:     if( Fgas_t(i) .lt. 0.0d0 ) call negfix( Fgas_t(i), 'Fgas_t(i)',
121:    &      0.0d0 )
122:
123:     Burn_d(i) = Burn_t(i)
124:
125:     Fplas_in_d(i) = Fuelf_t(i-1) * fin_pellet
126:
127:     if( Fplas_in_d(i) .lt. 0.0d0 ) call negfix( Fplas_in_d(i),
128:    &      'Fplas_in_d(i)', 0.0d0 )
129:
130:     Fuel_d(i) = Fplas_in_d(i) + Fgas_d(i-1) + Fplas_nb_d(i-1) +
131:    &      Fplas_prop_d(i-1)
132:
133:     Fex_d(i) = Fuel_d(i) - Burn_d(i) - ( d_Adiv_d(i-1) + d_Afw_d(i-1)
134:    &      ) - d_C_d(i-1) - d_D_d(i-1)
135:
136:     if( Fex_d(i) .lt. 0.0d0 ) call negfix( Fex_d(i), 'Fex_d(i)',
137:    &      0.0d0 )
138:
139:     Reac_p(i) = ap * fbu * Burn_t(i)
140:
141:     Fuel_p(i) = Fplas_in_p(i-1) + Xgas_p(i-1) * Fgas_tot(i-1) +
142:    &      Fplas_prop_p(i-1)
143:
144:     Fex_p(i) = Reac_p(i) + Fuel_p(i) - ( d_Adiv_p(i-1) + d_Afw_p(i-1)
145:    &      ) - d_C_p(i-1) - d_D_p(i-1)
146:
147:     if( Fex_p(i) .lt. 0.0d0 ) call negfix( Fex_p(i), 'Fex_p(i)',
148:    &      0.0d0 )
149:
150:     Fhe_ex(i) = Burn_t(i)
151:
152:     fmMVdiv = fm ( Medge ) * ai ( Temperature , t ) * Vdiv / ( 0.5 *
153:    &      3.016D0 )
154:
155:     fmMVfw = fm ( Medge ) * ai ( Temperature , t ) * Vfw / ( 0.5 *
156:    &      3.016D0 )
157:
158:     FFF = Fex_t(i) + Fex_d(i) + Fex_p(i)
159:
160:c    write(6,'(4(A,1PE12.5)))') 'FFF =',FFF,' Fex_t=',Fex_t(i),
161:c   &    ' Fex_d=',Fex_d(i),' Fex_p=',Fex_p(i)
162:
163:     if ( FFF .ne. 0.0d0 ) then
164:
165:       i_t(i) = Fex_t(i) / FFF
166:
167:       i_d(i) = Fex_d(i) / FFF
168:
169:       i_p(i) = Fex_p(i) / FFF
170:
171:     else
172:       i_t(i) = 0.0d0
173:
174:       i_d(i) = 0.0d0
175:
176:       i_p(i) = 0.0d0
177:
178:     endif
179:
180:     Adiv_t(i) = fmMVdiv * i_t(i) * fpp
181:
182:     Afw_t(i) = fmMVfw * i_t(i) * fpp
183:
184:     Adiv_d(i) = fmMVdiv * i_d(i) * fpp
185:
186:     Afw_d(i) = fmMVfw * i_d(i) * fpp
187:
188:     Adiv_p(i) = fmMVdiv * i_p(i) * fpp
189:
190:     Afw_p(i) = fmMVfw * i_p(i) * fpp
191:
192:     d_Adiv_t(i) = Adiv_t(i) - Adiv_t(i-1) - lambda_T * Adiv_t(i)
193:
194:     d_Adiv_d(i) = Adiv_d(i) - Adiv_d(i-1)
195:
```

```
196:     d_Adiv_p(i) = Adiv_p(i) - Adiv_p(i-1)
197:
198:     d_Afw_t(i) = Afw_t(i) - Afw_t(i-1) - lambda_T * Afw_t(i)
199:
200:     d_Afw_d(i) = Afw_d(i) - Afw_d(i-1)
201:
202:     d_Afw_p(i) = Afw_p(i) - Afw_p(i-1)
203:
204:     Tpfc_(i) = Adiv_t(i) + Afw_t(i) + C_t(i) + D_t(i)
205:
206:     Tpfc_m(i) = mu_A * ( Adiv_t(i) + Afw_t(i) ) + mu_C * C_t(i) +
207:    &      mu_D * D_t(i)
208:
209:     Dpfc_(i) = Adiv_d(i) + Afw_d(i) + C_d(i) + D_d(i)
210:
211:     Ppfc_(i) = Adiv_p(i) + Afw_p(i) + C_p(i) + D_p(i)
212:
213:     d_C_t(i) = ARcM * i_t(i) - lambda_T * C_t(i)
214:
215:     d_C_d(i) = ARcM * i_d(i)
216:
217:     d_C_p(i) = ARcM * i_p(i)
218:
219:     d_D_t(i) = EdMAd * i_t(i) - lambda_T * D_t(i)
220:
221:     d_D_d(i) = EdMAd * i_d(i)
222:
223:     d_D_p(i) = EdMAd * i_p(i)
224:*
225:* ... initial step is only for Plasma & fueling system
226:*
227:*if ( Initial_step.ne.0 ) then;
228:*  return ;
229:*endif ;
230:*
231:*#################################################################
232:*##(Vacuum)
233:*#################################################################
234:*
235:*
236:* Regeneration Cryopumps -------------------------------
237:*
238:     emrRSAp = emr * Rs / Area_p
239:c
240:     tps = 0.2d0/3600d0
241:c
242:c Tv( pump cycle time ) set
243:     T_pcycle = 2.0D0
244:     do ip = 1, NT_PUMP
245:        tvc = tmax*(ic-1)+tcycle+T_pcycle-T_pcycle/24*(ip-1)
246:        Tv(ip) = mod( tvc, T_pcycle )
247:*       if((T_pcycle - Tv(ip)) .lt. tps ) Tv(ip) = T_pcycle
248:        if(Tv(ip) .lt. tps ) Tv(ip) = T_pcycle
249:     end do
250:c    if( ic .lt. 5 )
251:c   &    write(6,'(A,1P24E12.5)') 'tv',(Tv(ip),ip=1,24)
252:c
253:     Tpv(i)   = 0.0D0
254:     Dpv(i)   = 0.0D0
255:     Ppv(i)   = 0.0D0
256:     Hepv(i)  = 0.0D0
257:     Rt_rem(i) = 0.0D0
258:     Rd_rem(i) = 0.0D0
259:     Rp_rem(i) = 0.0D0
260:     Rhe_rem(i) = 0.0D0
261:     Tp_tot(i) = 0.0D0
262:     Dp_tot(i) = 0.0D0
263:     Pp_tot(i) = 0.0D0
264:     Hep_tot(i) = 0.0D0
265:     do ip = 1, NT_PUMP
266:c
267:        if( Tv(ip).lt.(tpump+tps) ) then
268:          call ABXdelta( Fex_t(i)/npump, 0.0d0, Tpv_i(0,ip), i,
269:    &         delta_t, 'Tpv_i' , iabxctl )
270:c
271:          call ABXdelta( Fex_d(i)/npump, 0.0d0, Dpv_i(0,ip), i,
272:    &         delta_t, 'Dpv_i' , iabxctl )
273:c
274:          call ABXdelta( Fex_p(i)/npump, 0.0d0, Ppv_i(0,ip), i,
275:    &        delta_t, 'Ppv_i', iabxctl )
276:c
277:          call ABXdelta( Fhe_ex(i)/npump, 0.0d0, Hepv_i(0,ip), i,
278:    &        delta_t, 'Hepv_i', iabxctl )
279:
280:          Rt_rem_i(i,ip) = 0.d0
281:          Rd_rem_i(i,ip) = 0.d0
282:          Rp_rem_i(i,ip) = 0.d0
283:          Rhe_rem_i(i,ip) = 0.d0
284:
285:c
286:        else if( Tv(ip).lt.(tpump+5.0D0/60.0D0+tps) ) then
287:          call ABXdelta( 0.0d0, 0.0d0, Tpv_i(0,ip), i,
288:    &                delta_t, 'Tpv_i', iabxctl )
289:          call ABXdelta( 0.0d0, 0.0d0, Dpv_i(0,ip), i,
290:    &                delta_t, 'Dpv_i', iabxctl )
291:          call ABXdelta( 0.0d0, 0.0d0, Ppv_i(0,ip), i,
292:    &                delta_t, 'Ppv_i', iabxctl )
293:          call ABXdelta( 0.0d0, 0.0d0, Hepv_i(0,ip), i,
294:    &                delta_t, 'Hepv_i', iabxctl )
295:          Rt_rem_i(i,ip) = 0.0d0
296:          Rd_rem_i(i,ip) = 0.0d0
297:          Rp_rem_i(i,ip) = 0.0d0
298:          Rhe_rem_i(i,ip) = 0.0d0
299:
300:c
301:        else
302:c         Rt_rem_i(i,ip) = Tpv_i(i-1,ip) * emrRSAp * 2400.d0
303:c         Rd_rem_i(i,ip) = Dpv_i(i-1,ip) * emrRSAp * 2400.d0
304:c         Rp_rem_i(i,ip) = Ppv_i(i-1,ip) * emrRSAp * 2400.d0
305:c         Rhe_rem_i(i,ip) = Hepv_i(i-1,ip) * emrRSAp * 2400.d0
306:          Rt_rem_i(i,ip) = Tpv_i(i-1,ip) * emrRSAp
307:          Rd_rem_i(i,ip) = Dpv_i(i-1,ip) * emrRSAp
308:          Rp_rem_i(i,ip) = Ppv_i(i-1,ip) * emrRSAp
309:          Rhe_rem_i(i,ip) = Hepv_i(i-1,ip) * emrRSAp
```

```
310:          call ABXdelta( - Rt_rem_i(i,ip), 0.0d0, Tpv_i(0,ip), i,
311:    &                 delta_t, 'Tpv_i', iabxctl )
312:          if( Tpv_i(i,ip) .lt. 0.0d0 )
313:    &         call negfix( Tpv_i(i,ip), 'Tpv_i(i,ip)',0.0d0 )
314:
315:          call ABXdelta( - Rd_rem_i(i,ip), 0.0d0, Dpv_i(0,ip), i,
316:    &                 delta_t, 'Dpv_i', iabxctl )
317:          if( Dpv_i(i,ip) .lt. 0.0d0 )
318:    &         call negfix( Dpv_i(i,ip), 'Dpv_i(i,ip)',0.0d0 )
319:
320:          call ABXdelta( - Rp_rem_i(i,ip), 0.0d0, Ppv_i(0,ip), i,
321:    &                 delta_t, 'Ppv_i', iabxctl )
322:          if( Ppv_i(i,ip) .lt. 0.0d0 )
323:    &         call negfix( Ppv_i(i,ip), 'Ppv_i(i,ip)',0.0d0 )
324:
325:          call ABXdelta( - Rhe_rem_i(i,ip), 0.0d0, Hepv_i(0,ip), i,
326:    &                 delta_t, 'Hepv_i', iabxctl )
327:          if( Hepv_i(i,ip) .lt. 0.0d0 )
328:    &         call negfix( Hepv_i(i,ip), 'Hepv_i(i,ip)',0.0d0 )
329:
330:c
331:        end if
332:C***       if( Tv(ip) .lt. Tv_s(ip)) then
333:C***         Tp_reg_i(ip) = Tpv_i(i-1,ip)
334:C***         Dp_reg_i(ip) = Dpv_i(i-1,ip)
335:C***         Pp_reg_i(ip) = Ppv_i(i-1,ip)
336:C***         Hep_reg_i(ip) = Hepv_i(i-1,ip)
337:C***       end if
338:
339:        Tpv(i) = Tpv(i) + Tpv_i(i,ip)
340:        Dpv(i) = Dpv(i) + Dpv_i(i,ip)
341:        Ppv(i) = Ppv(i) + Ppv_i(i,ip)
342:        Hepv(i) = Hepv(i) + Hepv_i(i,ip)
343:        Rt_rem(i) = Rt_rem(i) + Rt_rem_i(i,ip)
344:        Rd_rem(i) = Rd_rem(i) + Rd_rem_i(i,ip)
345:        Rp_rem(i) = Rp_rem(i) + Rp_rem_i(i,ip)
346:        Rhe_rem(i) = Rhe_rem(i) + Rhe_rem_i(i,ip)
347:        Tp_tot(i) = Tp_tot(i) + Tpv_i(i,ip)
348:        Dp_tot(i) = Dp_tot(i) + Dpv_i(i,ip)
349:        Pp_tot(i) = Pp_tot(i) + Ppv_i(i,ip)
350:        Hep_tot(i) = Hep_tot(i) + Hepv_i(i,ip)
351:      end do
352:c     write(6,'(5(A,1PE12.5))') 'Tpv=',Tpv(i),
353:c   &  ' Tpv_i=',Tpv_i(i,1),' Dpv_i=',Dpv_i(i,1),
354:c   &  ' Ppv_i=',Ppv_i(i,1),' Hepv_i=',Hepv_i(i,1)
355:c
356:C***     do ip = 1, NT_PUMP
357:C***       Tv_s(ip) = Tv(ip)
358:C***     end do
359:c
360:c
361:*Tritium balance ------------------------------------
362:
363:      F(7) = ABX ( Fex_t(i-1) / npump , - emrRsAp , Tpv(i-1) )
364:c     call ABXdelta( Fex_t(i-1) / npump , - emrRsAp  , Tpv, i, delta_t,
365:c   &       'Tpv')
366:c
367:c     Rt_rem(i) = Tpv(i) * emrRsAp
368:c
369:c     Tp_tot(i) = Tpv(i) * npump
370:
371:      Tp_tot_m(i) = mu_v * Tp_tot(i)
372:* Deuterium Balance ------------------------------------
373:
374:      F(8) = ABX ( Fex_d(i-1) / npump , - emrRsAp , Dpv(i-1) )
375:c     call ABXdelta( Fex_d(i-1) / npump , - emrRsAp  , Dpv, i, delta_t,
376:c   &       'Dpv')
377:c
378:c     Rd_rem(i) = Dpv(i) * emrRsAp
379:c
380:c     Dp_tot(i) = Dpv(i) * npump
381:* Protium Balance ------------------------------------
382:
383:      F(9) = ABX ( Fex_p(i-1) / npump , - emrRsAp , Ppv(i-1) )
384:c     call ABXdelta( Fex_p(i-1) / npump , - emrRsAp  , Ppv, i, delta_t,
385:c   &       'Ppv')
386:c
387:c     Rp_rem(i) = Ppv(i) * emrRsAp
388:c
389:c     Pp_tot(i) = Ppv(i) * npump
390:* Helium Balance ------------------------------------
391:
392:      F(10) = ABX ( Fhe_ex(i-1) / npump , - emrRsAp , Hepv(i-1) )
393:c     call ABXdelta( Fhe_ex(i-1) / npump , - emrRsAp  , Hepv, i,
394:c   &       delta_t, 'Hepv')
395:c
396:c     Rhe_rem(i) = Hepv(i) * emrRsAp
397:c
398:c     Hep_tot(i) = Hepv(i) * npump
399:c
400:C***      Fv_out(i) = ( Rt_rem(i) + Rd_rem(i) + Rp_rem(i) + Rhe_rem(i) ) *
401:C***    &       npump
402:      Fv_out(i) = ( Rt_rem(i) + Rd_rem(i) + Rp_rem(i) + Rhe_rem(i) )
403:      if( Fv_out(i) .lt. 0.0d0 ) call negfix( Fv_out(i), 'Fv_out(i)',
404:     &       0.0d0 )
405:
406:      Fpuri_in(i) = Fv_out(i)
407:* Isotopic Fractions ------------------------------------------
408:
409:      FF = Rt_rem(i) + Rd_rem(i) + Rp_rem(i)
410:
411:      if ( FF .gt. 0.0d0 ) then
412:
413:        it_(i) = Rt_rem(i) / FF
414:
415:        id_(i) = Rd_rem(i) / FF
416:
417:        ip_(i) = Rp_rem(i) / FF
418:
419:      else
420:        write(6,'(3(A,1PE12.5)))') 'FF = 0.0  Rt_rem=',Rt_rem(i),
421:     &    ' Rd_rem=',Rd_rem(i),' Rp_rem=',Rp_rem(i)
422:
423:        it_(i) = 0.0d0
```

```
424:
425:        id_(i) = 0.0d0
426:
427:        ip_(i) = 0.0d0
428:
429:      endif
430:* To specify mole fractions of impurities in the exhaust stream
431:
432:      Fpure_(i) = Fv_out(i) * 0.98
433:
434:      Fimp_(i) = Fv_out(i) * 0.02
435:
436:      Fadj_puri(i) = Fv_out(i)
437:
438:      Fcq4_imp(i) = xcq4_ * Fimp_(i)
439:
440:      Fnq3_imp(i) = xnq3_ * Fimp_(i)
441:
442:      Fq2o_imp(i) = xq2o_ * Fimp_(i)
443:
444:      xh_imp(i) = xcq4_ + xnq3_ + xq2o_
445:
446:      xnhydro_(i) = 1 - 0.0136d0
447:
448:      Fnhydro_imp(i) = ( 1 - xh_imp(i) ) * Fimp_(i)
449:
450:      Fhydro_imp_w(i) = w1 * Fcq4_imp(i) + w2 * Fnq3_imp(i) + w3 *
451:     &       Fq2o_imp(i)
452:      if( Fhydro_imp_w(i) .lt. 0.0d0 ) call negfix( Fhydro_imp_w(i),
453:     &       'Fhydro_imp_w(i)', 0.0d0 )
454:
455:C=================================================================
456:C     Ft_pure(i) = Rt_rem(i) * npump - Fhydro_imp_w(i) * it_(i)
457:      Ft_pure(i) = Rt_rem(i) - Fhydro_imp_w(i) * it_(i)
458:      if( Ft_pure(i) .lt. 0.0d0 ) call negfix( Ft_pure(i),
459:     &       'Ft_pure(i)', 0.0d0 )
460:
461:C     Fd_pure(i) = Rd_rem(i) * npump - Fhydro_imp_w(i) * id_(i)
462:      Fd_pure(i) = Rd_rem(i) - Fhydro_imp_w(i) * id_(i)
463:      if( Fd_pure(i) .lt. 0.0d0 ) call negfix( Fd_pure(i),
464:     &       'Fd_pure(i)', 0.0d0 )
465:
466:C     Fp_pure(i) = Rp_rem(i) * npump - Fhydro_imp_w(i) * ip_(i)
467:      Fp_pure(i) = Rp_rem(i) - Fhydro_imp_w(i) * ip_(i)
468:      if( Fp_pure(i) .lt. 0.0d0 ) call negfix( Fp_pure(i),
469:     &       'Fp_pure(i)', 0.0d0 )
470:*
471:*#################################################################
472:*##(Fuel Clean-Up)
473:*#################################################################
474:*
475:*##.Flow into PURIFICATION
476:*
477:* .in purification mode
478:* Total Impurities:
479:
480:      F(11) = ABX ( ( xnhydro_(i) + xh_imp(i) ) * Fimp_(i-1) ,
481:     &       0.0d0 , Nimpt(i-1) )
482:      call ABXdelta( ( xnhydro_(i) + xh_imp(i) ) * Fimp_(i-1) ,
483:     &       0.0d0 , Nimpt, i, delta_t, 'Nimpt', iabxctl)
484:* Total Non-hydrogen-containing Impurities:
485:
486:      F(12) = ABX ( xnhydro_(i) * Fimp_(i-1) , 0.0d0 ,
487:     &       Nnhydro_imp(i-1) )
488:      call ABXdelta( xnhydro_(i) * Fimp_(i-1) , 0.0d0 ,
489:     &       Nnhydro_imp, i, delta_t, 'Nnhydro_imp', iabxctl)
490:* Hydrogen-containing Impurities:
491:
492:      F(13) = ABX ( xcq4_ * Fimp_(i-1) , 0.0d0 , Ncq4_imp(i-1) )
493:      call ABXdelta( xcq4_ * Fimp_(i-1) , 0.0d0 , Ncq4_imp, i,
494:     &       delta_t, 'Ncq4_imp', iabxctl)
495:
496:      F(14) = ABX ( xnq3_ * Fimp_(i-1) , 0.0d0 , Nnq3_imp(i-1) )
497:      call ABXdelta( xnq3_ * Fimp_(i-1) , 0.0d0 , Nnq3_imp, i,
498:     &       delta_t, 'Nnq3_imp', iabxctl)
499:
500:      F(15) = ABX ( xq2o_ * Fimp_(i-1) , 0.0d0 , Nq2o_imp(i-1) )
501:      call ABXdelta( xq2o_ * Fimp_(i-1) , 0.0d0 , Nq2o_imp, i,
502:     &       delta_t, 'Nq2o_imp', iabxctl)
503:c
504:* Hydrogen-containing Impurities:
505:
506:      f_hydro_imp_w(i) = w1 * xcq4_ + w2 * xnq3_ + w3 * xq2o_
507:*Tritium in Impurities:
508:
509:      F(16) = ABX ( f_hydro_imp_w(i) * it_(i-1) * Fimp_(i-1) ,
510:     &       0.0d0 , Timp_(i-1) )
511:      call ABXdelta( f_hydro_imp_w(i) * it_(i-1) * Fimp_(i-1) ,
512:     &       0.0d0 , Timp_, i, delta_t, 'Timp_', iabxctl)
513:*The mobilizable tritium is:
514:
515:      Timp_m(i) = mu_mob * Timp_(i)
516:*Deuterium in Impurities:
517:
518:      F(17) = ABX ( f_hydro_imp_w(i) * id_(i-1) * Fimp_(i-1) ,
519:     &       0.0d0 , Dimp_(i-1) )
520:      call ABXdelta( f_hydro_imp_w(i) * id_(i-1) * Fimp_(i-1) ,
521:     &       0.0d0 , Dimp_, i, delta_t, 'Dimp_', iabxctl)
522:*Protium in Impurities:
523:
524:      F(18) = ABX ( f_hydro_imp_w(i) * ip_(i-1) * Fimp_(i-1) ,
525:     &       0.0d0 , Pimp_(i-1) )
526:      call ABXdelta( f_hydro_imp_w(i) * ip_(i-1) * Fimp_(i-1) ,
527:     &       0.0d0 , Pimp_, i, delta_t, 'Pimp_', iabxctl)
528:*Total Hydrogen impurities:
529:
530:      Nhydro_imp(i) = Timp_(i) + Dimp_(i) + Pimp_(i)
531:c
532:c
533:c ....   tend set
534:c
535:      if( Nnq3_imp(i)+Nq2o_imp(i) .ge. Namsb_max .or.
536:     &   Ncq4_imp(i)+Nnhydro_imp(i) .ge. Ncmsb_max ) then
537:         tend = t
```

```
538:c       write(6,*) '======= tend set ',i
539:        if ( Nhydro_imp(i) .lt. 1.0d-5 ) then
540:          it_endj(i) = 0.0D0
541:          id_endj(i) = 0.0D0
542:          ip_endj(i) = 0.0D0
543:        else
544:          it_endj(i) = Timp_(i) / Nhydro_imp(i)
545:          id_endj(i) = Dimp_(i) / Nhydro_imp(i)
546:          ip_endj(i) = Pimp_(i) / Nhydro_imp(i)
547:        endif
548:        Ncq4_reg(i) = Ncq4_imp(i)
549:        Nnq3_reg(i) = Nnq3_imp(i)
550:        Nq2o_reg(i) = Nq2o_imp(i)
551:
552:        Nimpt(i) = 0.d0
553:        Nnhydro_imp(i) = 0.d0
554:        Timp_m(i) = 0.d0
555:
556:        Ncq4_imp(i) = 0.d0
557:        Nnq3_imp(i) = 0.d0
558:        Nq2o_imp(i) = 0.d0
559:        Timp_(i) = 0.d0
560:        Dimp_(i) = 0.d0
561:        Pimp_(i) = 0.d0
562:        Nhydro_imp(i) = 0.d0
563:
564:        Ns(i) = Namsb_max + Ncmsb_max
565:      else
566:        Ncq4_reg(i) = Ncq4_reg(i-1)
567:        Nnq3_reg(i) = Nnq3_reg(i-1)
568:        Nq2o_reg(i) = Nq2o_reg(i-1)
569:        it_endj(i) = it_endj(i-1)
570:        id_endj(i) = id_endj(i-1)
571:        ip_endj(i) = ip_endj(i-1)
572:      end if
573:c
574:c
575:*
576:* in Standby mode
577:*
578:
579:      Fh2_puri_in(i) = ( Ft_pure(i) + Fd_pure(i) + Fp_pure(i) ) / 2
580:
581:      FF = Ft_pure(i) + Fd_pure(i) + Fp_pure(i)
582:      Ft2_sub_in(i) = Ft_pure(i) * ( 0.5 + ( xnhydro_(i-1) +
583:     &      xh_imp(i-1) ) / FF * Fadj_puri(i) * Lhydro_ / Limp_ )
584:      if( Ft2_sub_in(i) .lt. 0.0d0 ) call negfix( Ft2_sub_in(i),
585:     &      'Ft2_sub_in(i)', 0.0d0 )
586:
587:      Fd2_sub_in(i) = Fd_pure(i) * ( 0.5 + ( xnhydro_(i-1) +
588:     &      xh_imp(i-1) ) / FF * Fadj_puri(i) * Lhydro_ / Limp_ )
589:      if( Fd2_sub_in(i) .lt. 0.0d0 ) call negfix( Fd2_sub_in(i),
590:     &      'Fd2_sub_in(i)', 0.0d0 )
591:
592:      Fp2_sub_in(i) = Fp_pure(i) * ( 0.5 + ( xnhydro_(i-1) +
593:     &      xh_imp(i-1) ) / FF * Fadj_puri(i) * Lhydro_ / Limp_ )
594:      if( Fp2_sub_in(i) .lt. 0.0d0 ) call negfix( Fp2_sub_in(i),
595:     &      'Fp2_sub_in(i)', 0.0d0 )
596:
597:      Fh2_sub_in(i) = Ft2_sub_in(i) + Fd2_sub_in(i) + Fp2_sub_in(i)
598:      if( Fh2_sub_in(i) .lt. 0.0d0 ) call negfix( Fh2_sub_in(i),
599:     &      'Fh2_sub_in(i)', 0.0d0 )
600:* Preloading with Hydrogen:
601:* tau_sb = Ns_max / Fh2_sub_in(t) ;
602:* Mole fractions
603:
604:      if ( Fh2_sub_in(i) .gt. 1.0d-10 ) then
605:
606:        xt_sub(i) = Ft2_sub_in(i) / Fh2_sub_in(i)
607:
608:        xd_sub(i) = Fd2_sub_in(i) / Fh2_sub_in(i)
609:
610:        xp_sub(i) = Fp2_sub_in(i) / Fh2_sub_in(i)
611:
612:      else
613:
614:        xt_sub(i) = 0.0d0
615:
616:        xd_sub(i) = 0.0d0
617:
618:        xp_sub(i) = 0.0d0
619:
620:      endif
621:
622:      if ( abs ( Fh2_sub_in(i) ) .lt. 1.0d-10 ) then
623:        write(6,'(A,1PE12.5))') 'Fh2_sub_in=',Fh2_sub_in(i)
624:
625:        tau_sb = 0.0d0
626:
627:      else
628:
629:        tau_sb = Ns_max / Fh2_sub_in(i)
630:
631:      endif
632:
633:      if ( abs ( tau_sb ) .ge. 1.0d-10 ) then
634:
635:        B = - 1.0d0 / tau_sb
636:
637:      else
638:c       write(6,'(A,1PE12.5))') 'tau_sb=',tau_sb
639:
640:        B = 0.0d0
641:
642:      endif
643:
644:C     write(6,*) 'tau_sb=',tau_sb,'  B=',B
645:      F(19) = ABX ( 0.0d0 , B , Ns(i-1) )
646:      if( t .ne. tend )
647:     &  call ABXdelta( 0.0d0, B, Ns, i, delta_t, 'Ns', iabxctl)
648:* Buildup of Hydrogen Molecules:
649:
650:      if ( abs ( tau_sb ) .ge. 1.0d-10 ) then
651:
```

```
652:        F(20) = ABX ( xt_sub(i-1) * Ns(i-1) / tau_sb , 0.0d0 ,
653:    &       T2_sub(i-1) )
654:        call ABXdelta( xt_sub(i-1) * Ns(i-1) / tau_sb , 0.0d0 ,
655:    &       T2_sub, i, delta_t, 'T2_sub', iabxctl)
656:
657:        F(21) = ABX ( xd_sub(i-1) * Ns(i-1) / tau_sb , 0.0d0 ,
658:    &       D2_sub(i-1) )
659:        call ABXdelta( xd_sub(i-1) * Ns(i-1) / tau_sb , 0.0d0 ,
660:    &       D2_sub, i, delta_t, 'D2_sub', iabxctl)
661:
662:        F(22) = ABX ( xp_sub(i-1) * Ns(i-1) / tau_sb , 0.0d0 ,
663:    &       P2_sub(i-1) )
664:        call ABXdelta( xp_sub(i-1) * Ns(i-1) / tau_sb , 0.0d0 ,
665:    &       P2_sub, i, delta_t, 'P2_sub', iabxctl)
666:
667:     else
668:
669:        F(20) = ABX ( 0.0d0 , 0.0d0 , T2_sub(i-1) )
670:        call ABXdelta( 0.0d0 , 0.0d0 , T2_sub, i, delta_t, 'T2_sub',
671:    &                iabxctl)
672:
673:        F(21) = ABX ( 0.0d0 , 0.0d0 , D2_sub(i-1) )
674:        call ABXdelta( 0.0d0 , 0.0d0 , D2_sub, i, delta_t, 'D2_sub',
675:    &                iabxctl)
676:
677:        F(22) = ABX ( 0.0d0 , 0.0d0 , P2_sub(i-1) )
678:        call ABXdelta( 0.0d0 , 0.0d0 , P2_sub, i, delta_t, 'P2_sub',
679:    &               iabxctl)
680:
681:     endif
682:
683:     Nhydro_sub(i) = T2_sub(i) + D2_sub(i) + P2_sub(i)
684:* Flow out from that is in Standby mode:
685:
686:     if ( abs ( tau_sb ) .ge. 1.0d-10 ) then
687:
688:        Ft_sub_out(i) = 2.0 * ( Ft2_sub_in(i) + xt_sub(i) * Ns(i) /
689:    &       tau_sb )
690:        if( Ft_sub_out(i) .lt. 0.0d0 ) call negfix( Ft_sub_out(i),
691:    &       'Ft_sub_out(i)', 0.0d0 )
692:
693:        Fd_sub_out(i) = 2.0 * ( Fd2_sub_in(i) + xd_sub(i) * Ns(i) /
694:    &       tau_sb )
695:        if( Fd_sub_out(i) .lt. 0.0d0 ) call negfix( Fd_sub_out(i),
696:    &       'Fd_sub_out(i)', 0.0d0 )
697:
698:        Fp_sub_out(i) = 2.0 * ( Fp2_sub_in(i) + xp_sub(i) * Ns(i) /
699:    &       tau_sb )
700:        if( Fp_sub_out(i) .lt. 0.0d0 ) call negfix( Fp_sub_out(i),
701:    &       'Fp_sub_out(i)', 0.0d0 )
702:
703:     else
704:
705:        Ft_sub_out(i) = 0.0d0
706:
707:        Fd_sub_out(i) = 0.0d0
708:
709:        Fp_sub_out(i) = 0.0d0
710:
711:     endif
712:
713:     Fh_sub_out(i) = Ft_sub_out(i) + Fd_sub_out(i) + Fp_sub_out(i)
714:     if( Fh_sub_out(i) .lt. 0.0d0 ) call negfix( Fh_sub_out(i),
715:    &       'Fh_sub_out(i)', 0.0d0 )
716:*
717:* .In Regeneration Mode
718:*
719:* Inventory during in Regeneration:
720:
721:     tpoint_cycle = t - Sigma_tend ( t,  tend )
722:     if ( tpoint_cycle .lt. treg_msieve ) then
723:
724:        Treg_out(i) = Timp_tend * ( 1 - tpoint_cycle / treg_msieve )
725:
726:     else
727:
728:        Treg_out(i) = 0D0
729:
730:     endif
731:* The vulnerable tritium inventory on the regenerating beds is:
732:
733:     Treg_out_m(i) = mu_reg_m * Treg_out(i)
734:* Evaluation of Hydrogen-containing Species:
735:
736:c    Ncq4_reg(i) = Ncq4_imp(j)
737:
738:c    Nnq3_reg(i) = Nnq3_imp(j)
739:
740:c    Nq2o_reg(i) = Nq2o_imp(j)
741:
742:     Ecq4_reg(i) = Ncq4_reg(i) / treg_msieve
743:
744:     Enq3_reg(i) = Nnq3_reg(i) / treg_msieve
745:
746:     Eq2o_reg(i) = Nq2o_reg(i) / treg_msieve
747:*
748:* Palladium Membrane Reactor:
749:*
750:
751:     Fhydro_pmem(i) = w1 * Ecq4_reg(i) + w2 * Enq3_reg(i) + w3 *
752:    &       Eq2o_reg(i)
753:* Inventories in the Palladium Membrane Reactor:
754:C=====================================================
755:cc   if ( j .lt. i .or. Nhydro_imp(j) .lt. 1.0d-5 ) then
756:c    if ( Nhydro_imp_tend .lt. 1.0d-5 ) then
757:
758:c       it_endj(i) = 0.0D0
759:
760:c       id_endj(i) = 0.0D0
761:
762:c       ip_endj(i) = 0.0D0
763:
764:c    else
765:
```

```
766:c       it_endj(i) = Timp_(i) / Nhydro_imp(i)
767:
768:c       id_endj(i) = Dimp_(i) / Nhydro_imp(i)
769:
770:c       ip_endj(i) = Pimp_(i) / Nhydro_imp(i)
771:
772:c     endif
773:
774:      F(23) = ABX ( it_endj(i-1) * Fhydro_pmem(i-1) , - 1.0d0 /
775:     &       tau_pmem , Tpmem(i-1) )
776:      call ABXdelta( it_endj(i-1) * Fhydro_pmem(i-1) , - 1.0d0 /
777:     &       tau_pmem  , Tpmem, i, delta_t, 'Tpmem',  iabxctl)
778:
779:      F(24) = ABX ( id_endj(i-1) * Fhydro_pmem(i-1) , - 1.0d0 /
780:     &       tau_pmem , Dpmem(i-1) )
781:      call ABXdelta( id_endj(i-1) * Fhydro_pmem(i-1) , - 1.0d0 /
782:     &       tau_pmem  , Dpmem, i, delta_t, 'Dpmem',  iabxctl)
783:
784:      F(25) = ABX ( ip_endj(i-1) * Fhydro_pmem(i-1) , - 1.0d0 /
785:     &       tau_pmem , Ppmem(i-1) )
786:      call ABXdelta( ip_endj(i-1) * Fhydro_pmem(i-1) , - 1.0d0 /
787:     &       tau_pmem  , Ppmem, i, delta_t, 'Ppmem',  iabxctl)
788:*
789:*
790:*  .Flow out from PURIFICATION
791:*  .to Isotope Separation System
792:*
793:
794:      if ( Tpmem(i) .lt. 1.0d-10 ) then
795:
796:        Ft_puri_out(i) = Ft_sub_out(i)
797:
798:      else
799:
800:        Ft_puri_out(i) = Ft_sub_out(i) + Tpmem(i) / tau_pmem
801:
802:      endif
803:      if( Ft_puri_out(i) .lt. 0.0d0 ) call negfix( Ft_puri_out(i),
804:     &       'Ft_puri_out(i)', 0.0d0 )
805:
806:      if ( Dpmem(i) .lt. 1.0d-10 ) then
807:
808:        Fd_puri_out(i) = Fd_sub_out(i)
809:
810:      else
811:
812:        Fd_puri_out(i) = Fd_sub_out(i) + Dpmem(i) / tau_pmem
813:
814:      endif
815:      if( Fd_puri_out(i) .lt. 0.0d0 ) call negfix( Fd_puri_out(i),
816:     &       'Fd_puri_out(i)', 0.0d0 )
817:
818:      if ( Ppmem(i) .lt. 1.0d-10 ) then
819:
820:        Fp_puri_out(i) = Fp_sub_out(i)
821:
822:      else
823:
824:        Fp_puri_out(i) = Fp_sub_out(i) + Ppmem(i) / tau_pmem
825:
826:      endif
827:      if( Fp_puri_out(i) .lt. 0.0d0 ) call negfix( Fp_puri_out(i),
828:     &       'Fp_puri_out(i)', 0.0d0 )
829:
830:      Fpuri_out(i) = Ft_puri_out(i) + Fd_puri_out(i) + Fp_puri_out(i)
831:      if( Fpuri_out(i) .lt. 0.0d0 ) call negfix( Fpuri_out(i),
832:     &       'Fpuri_out(i)', 0.0d0 )
833:*
834:*#################################################################
835:*##(Isotope Separation System)
836:*#################################################################
837:*
838:*  .Input stream to the ISS
839:*
840:
841:      Fiss_in(i) = Fbl_cool + Fbl_ + ( 1 - h1 - h3 ) * Fpuri_out(i) +
842:     &       Fragnb * Fiss_nb(i-1) + Fragiss_inj * Fiss_inj(i-1)
843:
844:      if( Fiss_in(i) .lt. 0.0d0 ) call negfix( Fiss_in(i),
845:     &       'Fiss_in(i)', 0.0d0 )
846:* .tritium flow
847:
848:      Fiss_in_t(i) = Fbl_cool * Xc_t + Fbl_ * Xbl_t + ( 1 - h1 - h3 ) *
849:     &       Fpuri_out(i) * Xiss_in_t(i-1) + Fragnb * Fiss_nb(i-1) *
850:     &       Xnb_t(i-1) + Fragiss_inj * Fiss_inj(i-1) * Xinj_t(i-1)
851:      if( Fiss_in_t(i) .lt. 0.0d0 ) call negfix( Fiss_in_t(i),
852:     &       'Fiss_in_t(i)', 0.0d0 )
853:* .deuteriumflow
854:
855:      Fiss_in_d(i) = Fbl_cool * Xc_d + Fbl_ * Xbl_d + ( 1 - h1 - h3 ) *
856:     &       Fpuri_out(i) * Xiss_in_d(i-1) + Fragnb * Fiss_nb(i-1) *
857:     &       Xnb_d(i-1) + Fragiss_inj * Fiss_inj(i-1) * Xinj_d(i-1)
858:      if( Fiss_in_d(i) .lt. 0.0d0 ) call negfix( Fiss_in_d(i),
859:     &       'Fiss_in_d(i)', 0.0d0 )
860:
861:      Xnb_t(i) = Xnb_cryo_t(i-1)
862:
863:      Xnb_d(i) = Xnb_cryo_d(i-1)
864:* Protium flow
865:
866:      Fiss_in_p(i) = Fiss_in(i) - Fiss_in_t(i) - Fiss_in_d(i)
867:      if( Fiss_in_p(i) .lt. 0.0d0 ) call negfix( Fiss_in_p(i),
868:     &       'Fiss_in_p(i)', 0.0d0 )
869:* mole fraction
870:C=============================================
871:      if ( Fiss_in(i) .ne. 0.0d0 ) then
872:
873:        Xiss_in_t(i) = Fiss_in_t(i) / Fiss_in(i)
874:
875:        Xiss_in_d(i) = Fiss_in_d(i) / Fiss_in(i)
876:
877:        Xiss_in_p(i) = 1 - Xiss_in_t(i) - Xiss_in_d(i)
878:
879:      else
```

```
880:        write(6,'(A,1PE12.5)') 'C======= Fiss_in=',Fiss_in(i)
881:
882:        Xiss_in_t(i) = 0.0d0
883:
884:        Xiss_in_d(i) = 0.0d0
885:
886:        Xiss_in_p(i) = 0.0d0
887:
888:      endif
889:
890:*
891:* .Isotope Separation System
892:*
893:*  overall balance
894:
895:      F(26) = ABX ( Fiss_in(i-1) , - 1.0d0 / tau_iss_ , Niss_(i-1) )
896:      call ABXdelta( Fiss_in(i-1) , - 1.0d0 / tau_iss_ , Niss_, i,
897:     &      delta_t, 'Niss_', iabxctl)
898:* tritium balance
899:
900:      F(27) = ABX ( Fiss_in_t(i-1) , - 1.0d0 / tau_iss_ , Tiss_(i-1) )
901:      call ABXdelta( Fiss_in_t(i-1) , - 1.0d0 / tau_iss_ , Tiss_, i,
902:     &      delta_t, 'Tiss_', iabxctl)
903:* mobilizable tritium
904:
905:      Tiss_m(i) = mu_iss_ * Tiss_(i)
906:* .deuterium balance
907:
908:      F(28) = ABX ( Fiss_in_d(i-1) , - 1.0d0 / tau_iss_ , Diss_(i-1) )
909:      call ABXdelta( Fiss_in_d(i-1) , - 1.0d0 / tau_iss_ , Diss_, i,
910:     &      delta_t, 'Diss_', iabxctl)
911:*
912:C=========================================
913:      if ( Niss_(i) .gt. 0.0 ) then
914:
915:        Xiss_t(i) = Tiss_(i) / Niss_(i)
916:
917:        Xiss_d(i) = Diss_(i) / Niss_(i)
918:
919:        Xiss_p(i) = 1.0d0 - Xiss_t(i) - Xiss_d(i)
920:
921:      else
922:        write(6,'(A,1PE12.5)') 'C========= Niss_=',Niss_(i)
923:
924:        Xiss_t(i) = 0.0d0
925:
926:        Xiss_d(i) = 0.0d0
927:
928:        Xiss_p(i) = 0.0d0
929:
930:      endif
931:* .protium balance
932:
933:      Piss_(i) = Xiss_p(i) * Niss_(i)
934:*
935:* III. exit flow from the ISS
936:*
937:*-----------------------------------------------------------
938:*  Solve the follwing equations:
939:*  Fiss_top_ + Fiss_mid_ + Fiss_bot_ = Niss_/tau_iss_
940:*  Fiss_top_*Xiss_top_t + Fiss_mid_*Xiss_mid_t + Fiss_bot_*Xiss_bot_t
941:*       = Tiss_/tau_iss_
942:*  Fiss_top_*Xiss_top_d + Fiss_mid_*Xiss_mid_d + Fiss_bot_*Xiss_bot_d
943:*       = Diss_/tau_iss_
944:*-----------------------------------------------------------
945:*
946:
947:      P1 = 1.0 / ( Xiss_mid_t * Xiss_bot_d + Xiss_bot_t * Xiss_top_d +
948:     &      Xiss_top_t * Xiss_mid_d - Xiss_mid_t * Xiss_top_d -
949:     &      Xiss_top_t * Xiss_bot_d - Xiss_bot_t * Xiss_mid_d )
950:
951:      A(1,1) = Xiss_top_t
952:      A(1,2) = Xiss_top_d
953:      A(1,3) = 0.0d0
954:      A(2,1) = Xiss_mid_t
955:      A(2,2) = Xiss_mid_d
956:      A(2,3) = 0.0d0
957:      A(3,1) = Xiss_bot_t
958:      A(3,2) = Xiss_bot_d
959:      A(3,3) = 0.0d0
960:      CA = condvar( 3, 3, A, 1.0/P1 )
961:      if( CA .gt. condn )  then
962:        write(66,*)  'WARNING  Fiss_top_ Fiss_mid_ Fiss_bot_'
963:        write(66,*)  '  CYCLE:',ic,'   time:',t*3600d0,' sec'
964:        write(66,'(A,6(1PE15.8),2(A,E15.8))')  '  aij = ',
965:     &    Xiss_top_t,Xiss_top_d,Xiss_mid_t,Xiss_mid_d,
966:     &    Xiss_bot_t,Xiss_bot_d,'   delta=',1.0/P1,
967:     &    '  condvar=',CA
968:      end if
969:      if( CA .gt. scondvar(1) )  then
970:        mcycle(1) = ic
971:        stime(1) = t
972:        scondvar(1) = CA
973:      end if
974:
975:      Fiss_top_(i) = P1 * ( ( Xiss_mid_t * Xiss_bot_d - Xiss_bot_t *
976:     &      Xiss_mid_d ) * Niss_(i) / tau_iss_ + ( Xiss_mid_d -
977:     &      Xiss_bot_d ) * Tiss_(i) / tau_iss_ + ( Xiss_bot_t -
978:     &      Xiss_mid_t ) * Diss_(i) / tau_iss_ )
979:      if( Fiss_top_(i) .lt. 0.0d0 ) call negfix( Fiss_top_(i),
980:     &      'Fiss_top_(i)', 0.0d0 )
981:
982:      Fiss_mid_(i) = P1 * ( ( Xiss_bot_t * Xiss_top_d - Xiss_top_t *
983:     &      Xiss_bot_d ) * Niss_(i) / tau_iss_ + ( Xiss_bot_d -
984:     &      Xiss_top_d ) * Tiss_(i) / tau_iss_ + ( Xiss_top_t -
985:     &      Xiss_bot_t ) * Diss_(i) / tau_iss_ )
986:      if( Fiss_mid_(i) .lt. 0.0d0 ) call negfix( Fiss_mid_(i),
987:     &      'Fiss_mid_(i)', 0.0d0 )
988:
989:      Fiss_bot_(i) = P1 * ( ( Xiss_top_t * Xiss_mid_d - Xiss_mid_t *
990:     &      Xiss_top_d ) * Niss_(i) / tau_iss_ + ( Xiss_top_d -
991:     &      Xiss_mid_d ) * Tiss_(i) / tau_iss_ + ( Xiss_mid_t -
992:     &      Xiss_top_t ) * Diss_(i) / tau_iss_ )
993:      if( Fiss_bot_(i) .lt. 0.0d0 ) call negfix( Fiss_bot_(i),
```

```
994:    &      'Fiss_bot_(i)', 0.0d0 )
995:*
996:*##############################################################
997:*##(Deuterium Neutral Beams)
998:*##############################################################
999:*
1000:* .flow to the Neutral beams
1001:*
1002:
1003:     W_nb(i) = 75D0 * fppower0 ( tcycle , t_burn , t_p_ru , t_p_rd
1004:    &                          , t_dwell )
1005:
1006:     Fplas_nb_d(i) = 3.6D3 / 1.6D0 * 1.0D19 * 1 / No * W_nb(i) / E_nb
1007:
1008:c==========================
1009:     if ( ( tcycle .lt. t_p_ru ) .or. ( tcycle .ge. tcycle_nb ) ) then
1010:
1011:       Crec(i) = 0.0D0
1012:
1013:     else
1014:
1015:       Crec(i) = 1.0D0
1016:
1017:     endif
1018:
1019:     Fin_rec_(i) = Crec(i-1) * Fnb_cryo(i-1)
1020:
1021:     Fiss_nb(i) = ( 1.0D0 - Crec(i-1) ) * Fnb_cryo(i-1)
1022:*
1023:* required flow from storage
1024:*
1025:c==================================================
1026:     if( Xsto_deu_d(i-1) .eq. 0.0d0 ) then
1027:       write(6,'(A,1PE12.5)')  'Xsto_deu_d=',Xsto_deu_d(i-1)
1028:       Fin_st_d(i) = 0.0d0
1029:     else
1030:       Fin_st_d(i) = ( Fplas_nb_d(i) / eta_nb_tot - kiss_mid_ *
1031:    &      Fiss_mid_(i) * Xiss_mid_d ) / Xsto_deu_d(i-1)
1032:     endif
1033:     if( Fin_st_d(i) .lt. 0.0d0 ) call negfix( Fin_st_d(i),
1034:    &      'Fin_st_d(i)', 0.0d0 )
1035:* III.Continuous regeneration cryopump
1036:*
1037:
1038:     F(29) = ABX ( fcryo_ * ( kiss_mid_ * Fiss_mid_(i-1) + Fin_st_d(i)
1039:    &      ) + Fbk_nb_(i-1) , - emrRsAp , Ncryo_(i-1) )
1040:     call ABXdelta( fcryo_ * ( kiss_mid_ * Fiss_mid_(i-1) +
1041:    &      Fin_st_d(i) ) + Fbk_nb_(i-1) , - emrRsAp , Ncryo_, i,
1042:    &      delta_t, 'Ncryo_', iabxctl)
1043:
1044:     Rnb_reg_(i) = Ncryo_(i) * emr * Rs / Area_p
1045:
1046:     F(30) = ABX ( fcryo_ * ( kiss_mid_ * Fiss_mid_(i-1) * Xiss_mid_t
1047:    &      + Fin_st_d(i) * Xsto_deu_t(i-1) ) + Fbk_nb_t(i-1) , -
1048:    &      emrRsAp , Tnb_cryo(i-1) )
1049:     call ABXdelta( fcryo_ * ( kiss_mid_ * Fiss_mid_(i-1) * Xiss_mid_t
1050:    &      + Fin_st_d(i) * Xsto_deu_t(i-1) ) + Fbk_nb_t(i-1) , -
1051:    &      emrRsAp , Tnb_cryo, i, delta_t, 'Tnb_cryo', iabxctl)
1052:
1053:     Tnb_cryo_mob(i) = mu_nb_mob * Tnb_cryo(i)
1054:
1055:     Rnb_reg_t(i) = Tnb_cryo(i) * emrRsAp
1056:
1057:     F(31) = ABX ( fcryo_ * ( kiss_mid_ * Fiss_mid_(i-1) * Xiss_mid_d
1058:    &      + Fin_st_d(i) * Xsto_deu_d(i-1) ) + Fbk_nb_d(i-1) -
1059:    &      Rnb_reg_d(i-1) , - emrRsAp , Dnb_cryo(i-1) )
1060:     call ABXdelta( fcryo_ * ( kiss_mid_ * Fiss_mid_(i-1) * Xiss_mid_d
1061:    &      + Fin_st_d(i) * Xsto_deu_d(i-1) ) + Fbk_nb_d(i-1) -
1062:    &      Rnb_reg_d(i-1) , - emrRsAp , Dnb_cryo, i, delta_t,
1063:    &      'Dnb_cryo', iabxctl)
1064:
1065:     Rnb_reg_d(i) = Dnb_cryo(i) * emrRsAp
1066:* mole fraction on the neutral beam cryopump:
1067:
1068:     if ( Ncryo_(i) .gt. 0.0d0 ) then
1069:
1070:       Xnb_cryo_t(i) = Tnb_cryo(i) / Ncryo_(i)
1071:
1072:       Xnb_cryo_d(i) = Dnb_cryo(i) / Ncryo_(i)
1073:
1074:       Xnb_cryo_p(i) = 1.0 - Xnb_cryo_d(i) - Xnb_cryo_t(i)
1075:
1076:       if ( Xnb_cryo_p(i) .lt. 0.0d0 ) then
1077:
1078:         Xnb_cryo_p(i) = 0.0d0
1079:
1080:       endif
1081:
1082:     else
1083:       write(6,'(A,1PE12.5)')  'C========== Ncryo_=',Ncryo_(i)
1084:
1085:       Xnb_cryo_t(i) = 0.0d0
1086:
1087:       Xnb_cryo_d(i) = 0.0d0
1088:
1089:       Xnb_cryo_p(i) = 0.0d0
1090:
1091:     endif
1092:* protium balance
1093:
1094:     Pnb_cryo(i) = Xnb_cryo_p(i) * Ncryo_(i)
1095:
1096:     Rnb_reg_p(i) = Pnb_cryo(i) * emr * Rs / Area_p
1097:*
1098:* .Tritium in dump
1099:*
1100:* .Tritium in the walls
1101:* total inventory:
1102:
1103:     F(32) = ABX ( fnb_wall * ( kiss_mid_ * Fiss_mid_(i-1) *
1104:    &      Xiss_mid_t + Fin_st_d(i-1) * Xsto_deu_t(i-1) ) , 0.0d0 ,
1105:    &      Tnb_wall(i-1) )
1106:     call ABXdelta( fnb_wall * ( kiss_mid_ * Fiss_mid_(i-1) *
1107:    &      Xiss_mid_t + Fin_st_d(i-1) * Xsto_deu_t(i-1) ) , 0.0d0 ,
```

```
1108:    &      Tnb_wall, i, delta_t, 'Tnb_wall', iabxctl)
1109:
1110:     Tnb_wall_mob(i) = mu_wall_ * Tnb_wall(i)
1111:*
1112:*  Backflow from plasma:
1113:*
1114:
1115:     Fbk_nb_(i) = 3 * 5.98D-21 * nedge_tot
1116:
1117:     Fbk_nb_t(i) = 3 * 5.98D-21 * nedge_t
1118:
1119:     Fbk_nb_d(i) = 3 * 5.98D-21 * nedge_d
1120:*
1121:*  Flow to plasma
1122:*
1123:*total flow to plasma
1124:*tritium flow to the plasma
1125:*deuterium flow to the plasma
1126:
1127:     Fplas_nb_(i) = ( kiss_mid_ * Fiss_mid_(i) + Fin_st_d(i) ) *
1128:    &       eta_nb_tot
1129:     if( Fplas_nb_(i) .lt. 0.0d0 ) call negfix( Fplas_nb_(i),
1130:    &       'Fplas_nb_(i)', 0.0d0 )
1131:
1132:     Fplas_nb_t(i) = ( kiss_mid_ * Fiss_mid_(i) * Xiss_mid_t +
1133:    &       Fin_st_d(i) * Xsto_deu_t(i-1) ) * eta_nb_tot
1134:     if( Fplas_nb_t(i) .lt. 0.0d0 ) call negfix( Fplas_nb_t(i),
1135:    &       'Fplas_nb_t(i)', 0.0d0 )
1136:* protium flow to the plasma
1137:
1138:     Fplas_nb_p(i) = Fplas_nb_(i) - Fplas_nb_t(i) - Fplas_nb_d(i)
1139:
1140:     if( Fplas_nb_p(i) .lt. 0.0d0 ) call negfix( Fplas_nb_p(i),
1141:    &       'Fplas_nb_p(i)', 0.0d0 )
1142:* .Exhaust flow from neutral beam cryopump
1143:
1144:     Fnb_cryo(i) = Rnb_reg_t(i) + Rnb_reg_d(i) + Rnb_reg_p(i)
1145:     if( Fnb_cryo(i) .lt. 0.0d0 ) call negfix( Fnb_cryo(i),
1146:    &       'Fnb_cryo(i)', 0.0d0 )
1147:*
1148:*##############################################################
1149:*##(Pellet injector)
1150:*##############################################################
1151:*
1152:*  II. The required flows
1153:
1154:     P8 = Fplas_in_t(i) / ( 1.0 - finj_err ) - Fsto_out_p(i-1) *
1155:    &       Xsto_pro_t(i-1) - h3 * Fpuri_out(i-1) * Xiss_in_t(i-1) -
1156:    &       linj_mid_iss * Fiss_mid_(i-1) * Xiss_mid_t - linj_bot_iss
1157:    &       * Fiss_bot_(i-1) * Xiss_bot_t
1158:
1159:     P9 = Fplas_in_d(i) / ( 1.0 - finj_err ) - Fsto_out_p(i-1) *
1160:    &       Xsto_pro_d(i-1) - h3 * Fpuri_out(i-1) * Xiss_in_d(i-1) -
1161:    &       linj_mid_iss * Fiss_mid_(i-1) * Xiss_mid_d - linj_bot_iss
1162:    &       * Fiss_bot_(i-1) * Xiss_bot_d
1163:
1164:     PP = Xsto_tri_t(i-1) * Xsto_deu_d(i-1) - Xsto_deu_t(i-1) *
1165:    &       Xsto_tri_d(i-1)
1166:
1167:     A(1,1) = Xsto_tri_t(i-1)
1168:     A(2,1) = Xsto_tri_d(i-1)
1169:     A(1,2) = Xsto_deu_t(i-1)
1170:     A(2,2) = Xsto_deu_d(i-1)
1171:     CA = condvar( 2, 3, A, PP )
1172:     if( CA .gt. condn ) then
1173:       write(66,*)  'WARNING  Fsto_out_t Fsto_out_d'
1174:     end if
1175:     if( CA .gt. scondvar(2) ) then
1176:       mcycle(2) = ic
1177:       stime(2) = t
1178:       scondvar(2) = CA
1179:     end if
1180:
1181:c==============================
1182:     if( PP .eq. 0.0d0 ) then
1183:       write(6,'(A,1PE12.5)') 'PP=',PP
1184:       Fsto_out_t(i) = 0.0d0
1185:       Fsto_out_d(i) = 0.0d0
1186:     else
1187:       Fsto_out_t(i) = (Xsto_deu_d(i-1)*P8 - Xsto_deu_t(i-1)*P9)
1188:    &       / PP
1189:       Fsto_out_d(i) = (-Xsto_tri_d(i-1)*P8 + Xsto_tri_t(i-1)*P9)
1190:    &       / PP
1191:     endif
1192:c    if( ic .lt. 2 )
1193:c   & write(6,'(6(A,1PE12.5))') 'Fsto_out_d=',Fsto_out_d(i),
1194:c   &  ' : Xsto_tri_d ',Xsto_tri_d(i-1),
1195:c   &  ' Xsto_tri_t',Xsto_tri_t(i-1),
1196:c   &  '  P8',P8,'  P9',P9,'  PP',PP
1197:     if( Fsto_out_t(i) .lt. 0.0d0 ) call negfix( Fsto_out_t(i),
1198:    &       'Fsto_out_t(i)', 0.0d0 )
1199:
1200:     if( Fsto_out_d(i) .lt. 0.0d0 ) call negfix( Fsto_out_d(i),
1201:    &       'Fsto_out_d(i)', 0.0d0 )
1202:*
1203:* Filling extruder:
1204:*
1205:
1206:     dNext_12 = h3 * Fpuri_out(i-1) + linj_bot_iss * Fiss_bot_(i-1) +
1207:    &       linj_mid_iss * Fiss_mid_(i-1) + Fsto_out_t(i-1) +
1208:    &       Fsto_out_d(i-1) + Fsto_out_p(i-1)
1209:
1210:     dText_12 = h3 * Fpuri_out(i-1) * Xiss_in_t(i-1) + linj_bot_iss *
1211:    &       Fiss_bot_(i-1) * Xiss_bot_t + linj_mid_iss *
1212:    &       Fiss_mid_(i-1) * Xiss_mid_t + Fsto_out_t(i-1) *
1213:    &       Xsto_tri_t(i-1) + Fsto_out_d(i-1) * Xsto_deu_t(i-1) +
1214:    &       Fsto_out_p(i-1) * Xsto_pro_t(i-1)
1215:
1216:     dDext_12 = h3 * Fpuri_out(i-1) * Xiss_in_d(i-1) + linj_bot_iss *
1217:    &       Fiss_bot_(i-1) * Xiss_bot_d + linj_mid_iss *
1218:    &       Fiss_mid_(i-1) * Xiss_mid_d + Fsto_out_t(i-1) *
1219:    &       Xsto_tri_d(i-1) + Fsto_out_d(i-1) * Xsto_deu_d(i-1) +
1220:    &       Fsto_out_p(i-1) * Xsto_pro_d(i-1)
1221:
```

```
1222:     F(33) = ABX ( dNext_12 - ( Fplas_in_t(i-1) + Fplas_in_d(i-1) +
1223:    &      Fplas_in_p(i-1) ) / ( 1 - finj_err ) , 0.0d0 ,
1224:    &      Next_12(i-1) )
1225:     call ABXdelta( dNext_12 - ( Fplas_in_t(i-1) + Fplas_in_d(i-1) +
1226:    &      Fplas_in_p(i-1) ) / ( 1 - finj_err ) , 0.0d0  , Next_12,
1227:    &      i, delta_t, 'Next_12', iabxctl)
1228:
1229:     F(34) = ABX ( dText_12 - Fplas_in_t(i-1) / ( 1 - finj_err ) ,
1230:    &      0.0d0 , Text_12(i-1) )
1231:     call ABXdelta( dText_12 - Fplas_in_t(i-1) / ( 1 - finj_err ) ,
1232:    &      0.0d0  , Text_12, i, delta_t, 'Text_12', iabxctl)
1233:
1234:     F(35) = ABX ( dDext_12 - Fplas_in_d(i-1) / ( 1 - finj_err ) ,
1235:    &      0.0d0 , Dext_12(i-1) )
1236:     call ABXdelta( dDext_12 - Fplas_in_d(i-1) / ( 1 - finj_err ) ,
1237:    &      0.0d0  , Dext_12, i, delta_t, 'Dext_12', iabxctl)
1238:
1239:     if ( Next_12(i) .ne. 0.0d0 ) then
1240:
1241:       Xext_12_t(i) = Text_12(i) / Next_12(i)
1242:
1243:       Xext_12_d(i) = Dext_12(i) / Next_12(i)
1244:
1245:       Xext_12_p(i) = 1.0 - Xext_12_t(i) - Xext_12_d(i)
1246:
1247:     else
1248:       write(6,'(A,1PE12.5)') 'C========= Next_12=',Next_12(i)
1249:
1250:       Xext_12_t(i) = 0.0d0
1251:
1252:       Xext_12_d(i) = 0.0d0
1253:
1254:       Xext_12_p(i) = 0.0d0
1255:
1256:     endif
1257:
1258:     Pext_12(i) = Xext_12_p(i) * Next_12(i)
1259:*
1260:* Total inventory in extruders:
1261:*
1262:
1263:     Next_tot(i) = Next_12(i) * 2.0
1264:
1265:     Text_tot(i) = Text_12(i) * 2.0
1266:
1267:     Dext_tot(i) = Dext_12(i) * 2.0
1268:
1269:     Pext_tot(i) = Pext_12(i) * 2.0
1270:*
1271:* The mobilizable tritium inventory is:
1272:*
1273:
1274:     Text_m(i) = mu_ext * Text_tot(i)
1275:
1276:     Nextru_(i) = Next_12(i)
1277:
1278:     if ( Next_12(i) .ne. 0.0d0 ) then
1279:
1280:       Xextru_t(i) = Text_12(i) / Next_12(i)
1281:
1282:       Xextru_d(i) = Dext_12(i) / Next_12(i)
1283:
1284:     else
1285:       write(6,'(A,1PE12.5)') 'Next_12=',Next_12(i)
1286:
1287:       Xextru_t(i) = 0.0d0
1288:
1289:       Xextru_d(i) = 0.0d0
1290:
1291:     endif
1292:
1293:     Xextru_p(i) = 1.0d0 - Xextru_t(i) - Xextru_d(i)
1294:*
1295:* Pellet Characteristics
1296:*
1297:
1298:     Vinj_t(i) = Xextru_t(i) * Nextru_(i) * 18.894D-6 / 2
1299:
1300:     Vinj_d(i) = Xextru_d(i) * Nextru_(i) * 20.085D-6 / 2
1301:
1302:     Vinj_p(i) = Xextru_p(i) * Nextru_(i) * 23.326D-6 / 2
1303:* total volume in extruder
1304:
1305:     Vinj_tot(i) = Vinj_t(i) + Vinj_d(i) + Vinj_p(i)
1306:* pellet volume
1307:* molar content of pellet
1308:c=========================================
1309:     if( Vinj_tot(i) .eq. 0.0d0 )  then
1310:       write(6,'(A,1PE12.5)')  'Vinj_tot=',Vinj_tot(i)
1311:       npellet_t(i) =  0.0d0
1312:       npellet_d(i) =  0.0d0
1313:       npellet_p(i) =  0.0d0
1314:     else
1315:       npellet_t(i) = Vinj_t(i) /
1316:    &        Vinj_tot(i) * Vpellet * 2 * kpellet_t
1317:       npellet_d(i) = Vinj_d(i) /
1318:    &        Vinj_tot(i) * Vpellet * 2 * kpellet_d
1319:       npellet_p(i) = Vinj_p(i) /
1320:    &        Vinj_tot(i) * Vpellet * 2 * kpellet_p
1321:     endif
1322:* repetition rate of injector
1323:c=========================================
1324:     if( npellet_t(i) .eq. 0.0d0 )  then
1325:       write(6,'(A,1PE12.5)')  'npellet_t=', npellet_t(i)
1326:       hrinj_ =  0.0d0
1327:     else
1328:       hrinj_ = Fplas_in_t(i) / (npellet_t(i)*(1-finj_err) )
1329:     endif
1330:c    write(6,'(3(A,E12.5))') 'hrinj_',hrinj_,
1331:c   & ' :Fplas_in_t',Fplas_in_t(i),' npellet_t',npellet_t(i)
1332:*
1333:*  ... Rinj_ in Hz ...
1334:
1335:     Rinj_(i) = hrinj_ / 3600d0
```

```
1336:*
1337:* protium to plasma from pellet:
1338:
1339:     Fplas_in_p(i) = hrinj_ * npellet_p(i) * ( 1 - finj_err )
1340:c    if( ic .lt.2 )
1341:c   & write(6,'(3(A,E12.5))') 'Fplas_in_p=',Fplas_in_p(i),
1342:c   & ' : hrinj_',hrinj_,' npellet_p',npellet_p(i)
1343:     if( Fplas_in_p(i) .lt. 0.0d0 ) call negfix( Fplas_in_p(i),
1344:    &       'Fplas_in_p(i)', 0.0d0 )
1345:* total flow to plasma from pellet:
1346:
1347:     Finj_in_tot(i) = Fplas_in_t(i) + Fplas_in_d(i) + Fplas_in_p(i)
1348:     if( Finj_in_tot(i) .lt. 0.0d0 ) call negfix( Finj_in_tot(i),
1349:    &       'Finj_in_tot(i)', 0.0d0 )
1350:*
1351:* .Backflow from plasma
1352:*
1353:
1354:     Fbk_plas_(i) = 3 * 5.98D-22 * nedge_tot
1355:
1356:     Fbk_plas_t(i) = 3 * 5.98D-22 * nedge_t
1357:
1358:     Fbk_plas_d(i) = 3 * 5.98D-22 * nedge_d
1359:
1360:     Fbk_plas_p(i) = 3 * 5.98D-22 * nedge_p
1361:*
1362:* Propellant requirements
1363:*
1364:
1365:     Finj_prop(i) = nprop * hrinj_ / 3600d0
1366:c    if( ic .lt. 2 )
1367:c   & write(6,'(2(A,E12.5))') 'Finj_prop=',Finj_prop(i),
1368:c   & ' : hrinj_p',hrinj_
1369:     if( Finj_prop(i) .lt. 0.0d0 ) call negfix( Finj_prop(i),
1370:    &       'Finj_prop(i)', 0.0d0 )
1371:
1372:     Fsto_out_p(i) = Finj_prop(i) - liss_p * Fiss_top_(i-1)
1373:c=========================================
1374:c    if( Finj_prop(i) .eq. 0.0d0 ) then
1375:c       write(6,'(A,1PE12.5)') 'Finj_prop=', Finj_prop(i)
1376:c    endif
1377:
1378:     if ( Fsto_out_p(i) .gt. 0 ) then
1379:       if( Finj_prop(i) .eq. 0.0d0 ) then
1380:         Xprop_t(i) = Xiss_top_t
1381:         Xprop_d(i) = Xiss_top_d
1382:       else
1383:         Xprop_t(i) = ( Fsto_out_p(i-1) * Xsto_pro_t(i-1) + liss_p *
1384:    &      Fiss_top_(i-1) * Xiss_top_t ) / Finj_prop(i)
1385:         Xprop_d(i) = ( Fsto_out_p(i-1) * Xsto_pro_d(i-1) + liss_p *
1386:    &      Fiss_top_(i-1) * Xiss_top_d ) / Finj_prop(i)
1387:       endif
1388:     else
1389:
1390:       Fsto_out_p(i) = 0.0d0
1391:
1392:       Xprop_t(i) = Xiss_top_t
1393:
1394:       Xprop_d(i) = Xiss_top_d
1395:
1396:     endif
1397:
1398:     Xprop_p(i) = 1 - Xprop_t(i) - Xprop_d(i)
1399:*
1400:* propellant flow to plasma
1401:*
1402:
1403:     Fplas_prop(i) = ( 1 - fcap_prop ) * Finj_prop(i)
1404:     if( Fplas_prop(i) .lt. 0.0d0 ) call negfix( Fplas_prop(i),
1405:    &       'Fplas_prop(i)', 0.0d0 )
1406:
1407:     Fplas_prop_t(i) = Xprop_t(i) * Fplas_prop(i)
1408:     if( Fplas_prop_t(i) .lt. 0.0d0 ) call negfix( Fplas_prop_t(i),
1409:    &       'Fplas_prop_t(i)', 0.0d0 )
1410:
1411:     Fplas_prop_d(i) = Xprop_d(i) * Fplas_prop(i)
1412:     if( Fplas_prop_d(i) .lt. 0.0d0 ) call negfix( Fplas_prop_d(i),
1413:    &       'Fplas_prop_d(i)', 0.0d0 )
1414:
1415:     Fplas_prop_p(i) = Xprop_p(i) * Fplas_prop(i)
1416:     if( Fplas_prop_p(i) .lt. 0.0d0 ) call negfix( Fplas_prop_p(i),
1417:    &       'Fplas_prop_p(i)', 0.0d0 )
1418:*
1419:* Exhaust flow from pellet injector
1420:*
1421:
1422:     Finj_exh(i) = ( ( Fplas_in_t(i) + Fplas_in_d(i) ) / ( 1 -
1423:    &       finj_err ) + Fplas_in_p(i) ) * finj_err + Finj_prop(i) *
1424:    &       fcap_prop + Fbk_plas_(i)
1425:     if( Finj_exh(i) .lt. 0.0d0 ) call negfix( Finj_exh(i),
1426:    &       'Finj_exh(i)', 0.0d0 )
1427:
1428:     Finj_exh_t(i) = Fplas_in_t(i) * finj_err / ( 1 - finj_err ) +
1429:    &       Finj_prop(i) * Xprop_t(i) * fcap_prop + Fbk_plas_t(i)
1430:     if( Finj_exh_t(i) .lt. 0.0d0 ) call negfix( Finj_exh_t(i),
1431:    &       'Finj_exh_t(i)', 0.0d0 )
1432:
1433:     Finj_exh_d(i) = Fplas_in_d(i) * finj_err / ( 1 - finj_err ) +
1434:    &       Finj_prop(i) * Xprop_d(i) * fcap_prop + Fbk_plas_d(i)
1435:     if( Finj_exh_d(i) .lt. 0.0d0 ) call negfix( Finj_exh_d(i),
1436:    &       'Finj_exh_d(i)', 0.0d0 )
1437:*
1438:* The flow directed to the ISS is given by:
1439:*
1440:
1441:     Fiss_inj(i) = ( 1 - fnon_iss ) * Finj_exh(i)
1442:     if( Fiss_inj(i) .lt. 0.0d0 ) call negfix( Fiss_inj(i),
1443:    &       'Fiss_inj(i)', 0.0d0 )
1444:*
1445:* recycle flow:
1446:*
1447:
1448:     Finj_rec(i) = fnon_iss * Finj_exh(i)
1449:     if( Finj_rec(i) .lt. 0.0d0 ) call negfix( Finj_rec(i),
```

```
1450:    &       'Finj_rec(i)', 0.0d0 )
1451:
1452:     if ( Finj_exh(i) .ne. 0.0d0 ) then
1453:
1454:       Xinj_t(i) = Finj_exh_t(i) / Finj_exh(i)
1455:
1456:       Xinj_d(i) = Finj_exh_d(i) / Finj_exh(i)
1457:
1458:       Xinj_p(i) = 1.0 - Xinj_t(i) - Xinj_d(i)
1459:
1460:     else
1461:        write(6,'(A,1PE12.5)')  'C======= Finj_exh=', Finj_exh(i)
1462:
1463:       Xinj_t(i) = 0.0d0
1464:
1465:       Xinj_d(i) = 0.0d0
1466:
1467:       Xinj_p(i) = 0.0d0
1468:
1469:     endif
1470:*
1471:*##############################################################
1472:*##(Gas Puffer)
1473:*##############################################################
1474:*
1475:
1476:     Fgas_d(i) = Fgas_t(i)
1477:*-------------------------------------------------------------
1478:*
1479:*  Fgas_in_tri(t) & Fgas_in_deu(t) are determined by solving
1480:*  the following equation.
1481:*
1482:* Fgas_t = Fgas_in_tri(t) * Xsto_tri_t + Fgas_in_deu(t) * Xsto_deu_t
1483:*      + h1 * Fpuri_out(t0) * Xiss_in_t
1484:*      + mgas_mid_iss * Fiss_mid_(t0) * Xiss_mid_t
1485:*      + mgas_bot_iss * Fiss_bot_(t0) * Xiss_bot_t ;
1486:*
1487:* Fgas_d = Fgas_in_tri(t) * Xsto_tri_d + Fgas_in_deu(t) * Xsto_deu_d
1488:*      + h1 * Fpuri_out(t0) * Xiss_in_d
1489:*      + mgas_mid_iss * Fiss_mid_(t0) * Xiss_mid_d
1490:*      + mgas_bot_iss * Fiss_bot_(t0) * Xiss_bot_d ;
1491:*
1492:*-------------------------------------------------------------
1493:*   ... solution ...
1494:*
1495:
1496:     B1 = Fgas_t(i) - ( h1 * Fpuri_out(i-1) * Xiss_in_t(i-1) +
1497:    &      mgas_mid_iss * Fiss_mid_(i-1) * Xiss_mid_t + mgas_bot_iss
1498:    &      * Fiss_bot_(i-1) * Xiss_bot_t )
1499:
1500:     B2 = Fgas_d(i) - ( h1 * Fpuri_out(i-1) * Xiss_in_d(i-1) +
1501:    &      mgas_mid_iss * Fiss_mid_(i-1) * Xiss_mid_d + mgas_bot_iss
1502:    &      * Fiss_bot_(i-1) * Xiss_bot_d )
1503:
1504:     DD = Xsto_tri_t(i-1) * Xsto_deu_d(i-1) - Xsto_deu_t(i-1) *
1505:    &      Xsto_tri_d(i-1)
1506:
1507:     A(1,1) = Xsto_tri_t(i-1)
1508:     A(2,1) = Xsto_tri_d(i-1)
1509:     A(1,2) = Xsto_deu_t(i-1)
1510:     A(2,2) = Xsto_deu_d(i-1)
1511:     CA = condvar( 2, 3, A, DD )
1512:     if( CA .gt. condn )  then
1513:       write(66,*)  'WARNING  Fgas_in_tri Fgas_in_deu'
1514:       write(66,'(A,I4,A,F10.0,A,A,4(1PE15.8),A,E15.8)')
1515:    &      ' CYCLE:',ic,'  time:',t*3600d0,' sec',
1516:    &      '  aij = ',Xsto_tri_t(i-1),Xsto_tri_d(i-1),
1517:    &      Xsto_deu_t(i-1),Xsto_deu_d(i-1),' condvar=',CA
1518:     end if
1519:     if( CA .gt. scondvar(3) )  then
1520:       mcycle(3) = ic
1521:       stime(3) = t
1522:       scondvar(3) = CA
1523:     end if
1524:
1525:     if ( abs ( DD ) .gt. 1.0d-10 ) then
1526:
1527:       Fgas_in_tri(i) = ( B1 * Xsto_deu_d(i-1) - B2 * Xsto_deu_t(i-1)
1528:    &      ) / DD
1529:       if( Fgas_in_tri(i) .lt. 0.0d0 ) call negfix( Fgas_in_tri(i),
1530:    &      'Fgas_in_tri(i)', 0.0d0 )
1531:
1532:       Fgas_in_deu(i) = ( B2 * Xsto_tri_t(i-1) - B1 * Xsto_tri_d(i-1)
1533:    &      ) / DD
1534:      if( Fgas_in_deu(i) .lt. 0.0d0 ) call negfix( Fgas_in_deu(i),
1535:    &      'Fgas_in_deu(i)', 0.0d0 )
1536:
1537:     else
1538:       write(6,'(A,1Pe12.5)') 'DD=', DD
1539:
1540:c=========================================
1541:       if( Xsto_tri_t(i-1) .eq. 0.0d0 ) then
1542:         write(6,'(A,1Pe12.5)') 'Xsto_tri_t=', Xsto_tri_t(i-1)
1543:         Fgas_in_tri(i) = 0.0d0
1544:       else
1545:       Fgas_in_tri(i) = ( Fgas_t(i) - ( Fgas_in_deu(i-1) *
1546:    &      Xsto_deu_t(i-1) + h1 * Fpuri_out(i-1) * Xiss_in_t(i-1) +
1547:    &      mgas_mid_iss * Fiss_mid_(i-1) * Xiss_mid_t + mgas_bot_iss
1548:    &      * Fiss_bot_(i-1) * Xiss_bot_t ) ) / Xsto_tri_t(i-1)
1549:       endif
1550:
1551:       if( Xsto_deu_d(i-1) .eq. 0.0d0 ) then
1552:         write(6,'(A,1Pe12.5)') 'Xsto_deu_d=', Xsto_deu_d(i-1)
1553:         Fgas_in_deu(i) = 0.0d0
1554:       else
1555:       Fgas_in_deu(i) = ( Fgas_d(i) - ( Fgas_in_tri(i-1) *
1556:    &      Xsto_tri_d(i-1) + h1 * Fpuri_out(i-1) * Xiss_in_d(i-1) +
1557:    &      mgas_mid_iss * Fiss_mid_(i-1) * Xiss_mid_d + mgas_bot_iss
1558:    &      * Fiss_bot_(i-1) * Xiss_bot_d ) ) / Xsto_deu_d(i-1)
1559:       endif
1560:
1561:     endif
1562:*
1563:* .Overall balance
```

```
1564:*
1565:
1566:     F(36) = ABX ( - Fgas_tot(i-1) + h1 * Fpuri_out(i-1) +
1567:    &      mgas_mid_iss * Fiss_mid_(i-1) + mgas_bot_iss *
1568:    &      Fiss_bot_(i-1) + Fgas_in_tri(i-1) + Fgas_in_deu(i-1) ,
1569:    &      0.0d0 , Ngas_tot(i-1) )
1570:     call ABXdelta( - Fgas_tot(i-1) + h1 * Fpuri_out(i-1) +
1571:    &      mgas_mid_iss * Fiss_mid_(i-1) + mgas_bot_iss *
1572:    &      Fiss_bot_(i-1) + Fgas_in_tri(i-1) + Fgas_in_deu(i-1) ,
1573:    &      0.0d0  , Ngas_tot, i, delta_t, 'Ngas_tot', iabxctl)
1574:* .tritium balance
1575:
1576:     F(37) = ABX ( - Fgas_t(i-1) + h1 * Fpuri_out(i-1) *
1577:    &      Xiss_in_t(i-1) + mgas_mid_iss * Fiss_mid_(i-1) *
1578:    &      Xiss_mid_t + mgas_bot_iss * Fiss_bot_(i-1) * Xiss_bot_t +
1579:    &      Fgas_in_tri(i-1) * Xsto_tri_t(i-1) + Fgas_in_deu(i-1) *
1580:    &      Xsto_deu_t(i-1) , 0.0d0 , Tgas_(i-1) )
1581:     call ABXdelta( - Fgas_t(i-1) + h1 * Fpuri_out(i-1) *
1582:    &      Xiss_in_t(i-1) + mgas_mid_iss * Fiss_mid_(i-1) *
1583:    &      Xiss_mid_t + mgas_bot_iss * Fiss_bot_(i-1) * Xiss_bot_t +
1584:    &      Fgas_in_tri(i-1) * Xsto_tri_t(i-1) + Fgas_in_deu(i-1) *
1585:    &      Xsto_deu_t(i-1) , 0.0d0  , Tgas_, i, delta_t, 'Tgas_',
1586:    &      iabxctl )
1587:
1588:* mobilizable tritium:
1589:
1590:     Tgas_mob(i) = mu_gas * Tgas_(i)
1591:* .deuterium balance
1592:
1593:     F(38) = ABX ( - Fgas_d(i-1) + h1 * Fpuri_out(i-1) *
1594:    &      Xiss_in_d(i-1) + mgas_mid_iss * Fiss_mid_(i-1) *
1595:    &      Xiss_mid_d + mgas_bot_iss * Fiss_bot_(i-1) * Xiss_bot_d +
1596:    &      Fgas_in_tri(i-1) * Xsto_tri_d(i-1) + Fgas_in_deu(i-1) *
1597:    &      Xsto_deu_d(i-1) , 0.0d0 , Dgas_(i-1) )
1598:     call ABXdelta( - Fgas_d(i-1) + h1 * Fpuri_out(i-1) *
1599:    &      Xiss_in_d(i-1) + mgas_mid_iss * Fiss_mid_(i-1) *
1600:    &      Xiss_mid_d + mgas_bot_iss * Fiss_bot_(i-1) * Xiss_bot_d +
1601:    &      Fgas_in_tri(i-1) * Xsto_tri_d(i-1) + Fgas_in_deu(i-1) *
1602:    &      Xsto_deu_d(i-1) , 0.0d0  , Dgas_, i, delta_t, 'Dgas_',
1603:    &      iabxctl)
1604:
1605:* mole fraction:
1606:c================================
1607:     if( Ngas_tot(i) .eq. 0.0d0 ) then
1608:       write(6,'(A,1PE12.5)')  'Ngas_tot=', Ngas_tot(i)
1609:       Xgas_t(i) = 0.0d0
1610:       Xgas_d(i) = 0.0d0
1611:     else
1612:       Xgas_t(i) = Tgas_(i) / Ngas_tot(i)
1613:
1614:       Xgas_d(i) = Dgas_(i) / Ngas_tot(i)
1615:     endif
1616:
1617:     Xgas_p(i) = 1 - Xgas_t(i) - Xgas_d(i)
1618:
1619:* .protium balance
1620:
1621:     Pgas_(i) = Xgas_p(i) * Ngas_tot(i)
1622:* total flow to plasma
1623:c================================
1624:     if( ( Xgas_t(i) + Xgas_d(i) ) .eq. 0.0d0 ) then
1625:        write(6,'(2(A,1PE12.5))') 'Xgas_t=',Xgas_t(i),
1626:    &        ' Xgas_d=',Xgas_d(i)
1627:        Fgas_tot(i) = 0.0d0
1628:     else
1629:        Fgas_tot(i) = ( Fgas_t(i) + Fgas_d(i) )
1630:    &                / ( Xgas_t(i) + Xgas_d(i) )
1631:     endif
1632:
1633:     if( Fgas_tot(i) .lt. 0.0d0 ) call negfix( Fgas_tot(i),
1634:    &       'Fgas_tot(i)', 0.0d0 )
1635:*
1636:* The total flow to be puffered into the plasma includes
1637:* an amount of pellet injector propellant:
1638:*
1639:
1640:     Fpuff_plas(i) = Fgas_tot(i) + Fplas_prop_t(i) + Fplas_prop_d(i) +
1641:    &       Fplas_prop_p(i)
1642:     if( Fpuff_plas(i) .lt. 0.0d0 ) call negfix( Fpuff_plas(i),
1643:    &       'Fpuff_plas(i)', 0.0d0 )
1644:*
1645:*##############################################################
1646:*###( Storage and external supply)
1647:*##############################################################
1648:*
1649:*  ---------tritium rich storage
1650:*
1651:
1652:     Fsto_tri_in(i) = ( 1 - linj_bot_iss - mgas_bot_iss ) *
1653:    &       Fiss_bot_(i-1)
1654:     if( Fsto_tri_in(i) .lt. 0.0d0 ) call negfix( Fsto_tri_in(i),
1655:    &       'Fsto_tri_in(i)', 0.0d0 )
1656:
1657:     Fsto_tri_out(i) = Fsto_out_t(i-1) + Fgas_in_tri(i-1)
1658:
1659:     if( Fsto_tri_out(i) .lt. 0.0d0 ) call negfix( Fsto_tri_out(i),
1660:    &       'Fsto_tri_out(i)', 0.0d0 )
1661:* overall balance:
1662:
1663:     F(39) = ABX ( Fsto_tri_in(i-1) - Fsto_tri_out(i-1) , 0.0d0 ,
1664:    &       Nsto_tri_(i-1) )
1665:     call ABXdelta( Fsto_tri_in(i-1) - Fsto_tri_out(i-1) , 0.0d0 ,
1666:    &       Nsto_tri_, i, delta_t, 'Nsto_tri_', iabxctl)
1667:* total tritium:
1668:c=====================================
1669:     if( Nsto_tri_(i-1) .eq. 0.0d0 ) then
1670:c        write(6,*) 'Nsto_tri_ ',Nsto_tri_(i-1)
1671:         F(40) = ABX ( Fsto_tri_in(i-1) * Xiss_bot_t , 0.0d0,
1672:    &                Tsto_tri_(i-1) )
1673:         call ABXdelta( Fsto_tri_in(i-1) * Xiss_bot_t ,  0.0d0,
1674:    &             Tsto_tri_, i, delta_t, 'Tsto_tri_', iabxctl)
1675:     else
1676:        F(40) = ABX ( Fsto_tri_in(i-1) * Xiss_bot_t ,
1677:    &        - Fsto_tri_out(i-1)  / Nsto_tri_(i-1) ,
```

```
1678:    &       Tsto_tri_(i-1) )
1679:       call ABXdelta( Fsto_tri_in(i-1) * Xiss_bot_t ,
1680:    &      - Fsto_tri_out(i-1) / Nsto_tri_(i-1)  , Tsto_tri_, i,
1681:    &      delta_t, 'Tsto_tri_', iabxctl)
1682:     endif
1683:* deuterium balance:
1684:
1685:     if( Nsto_tri_(i-1) .eq. 0.0d0 ) then
1686:       F(41) = ABX ( Fsto_tri_in(i-1) * Xiss_bot_d ,  0.0d0,
1687:    &      Dsto_tri_(i-1) )
1688:       call ABXdelta( Fsto_tri_in(i-1) * Xiss_bot_d ,  0.0d0,
1689:    &      Dsto_tri_, i, delta_t, 'Dsto_tri_', iabxctl)
1690:     else
1691:       F(41) = ABX ( Fsto_tri_in(i-1) * Xiss_bot_d ,
1692:    &      - Fsto_tri_out(i-1)/ Nsto_tri_(i-1) , Dsto_tri_(i-1) )
1693:       call ABXdelta( Fsto_tri_in(i-1) * Xiss_bot_d ,
1694:    &      - Fsto_tri_out(i-1) / Nsto_tri_(i-1)  , Dsto_tri_, i,
1695:    &      delta_t, 'Dsto_tri_', iabxctl)
1696:     endif
1697:*
1698:* --------- Deuterium rich storage
1699:*
1700:
1701:     Fsto_deu_in(i) = ( 1 - kiss_mid_ - linj_mid_iss - mgas_mid_iss )
1702:    &      * Fiss_mid_(i)
1703:     if( Fsto_deu_in(i) .lt. 0.0d0 ) call negfix( Fsto_deu_in(i),
1704:    &      'Fsto_deu_in(i)', 0.0d0 )
1705:
1706:     Fsto_deu_out(i) = Fsto_out_d(i-1) + Fgas_in_deu(i-1) +
1707:    &      Fin_st_d(i-1)
1708:
1709:     if( Fsto_deu_out(i) .lt. 0.0d0 ) call negfix( Fsto_deu_out(i),
1710:    &      'Fsto_deu_out(i)', 0.0d0 )
1711:* overall balance:
1712:
1713:     F(42) = ABX ( Fsto_deu_in(i-1) - Fsto_deu_out(i-1) , 0.0d0 ,
1714:    &      Nsto_deu_(i-1) )
1715:     call ABXdelta( Fsto_deu_in(i-1) - Fsto_deu_out(i-1) , 0.0d0  ,
1716:    &      Nsto_deu_, i, delta_t, 'Nsto_deu_', iabxctl)
1717:* total tritium:
1718:c=====================================
1719:     if( Nsto_deu_(i-1) .eq. 0.0d0 ) then
1720:        write(6,'(A,1Pe12.5)') 'Nsto_deu_=',Nsto_deu_(i-1)
1721:       F(43) = ABX ( Fsto_deu_in(i-1) * Xiss_mid_t , 0.0d0
1722:    &       , Tsto_deu_(i-1) )
1723:       call ABXdelta( Fsto_deu_in(i-1) * Xiss_mid_t , 0.0d0
1724:    &       , Tsto_deu_, i, delta_t, 'Tsto_deu_', iabxctl)
1725:     else
1726:       F(43) = ABX ( Fsto_deu_in(i-1) * Xiss_mid_t ,
1727:    &      - Fsto_deu_out(i-1) / Nsto_deu_(i-1) , Tsto_deu_(i-1) )
1728:       call ABXdelta( Fsto_deu_in(i-1) * Xiss_mid_t ,
1729:    &      - Fsto_deu_out(i-1) / Nsto_deu_(i-1)  , Tsto_deu_, i,
1730:    &      delta_t, 'Tsto_deu_', iabxctl)
1731:     endif
1732:* deuterium balance:
1733:
1734:     if( Nsto_deu_(i-1) .eq. 0.0d0 ) then
1735:       F(44) = ABX ( Fsto_deu_in(i-1) * Xiss_bot_d ,  0.0d0 ,
1736:    &      Dsto_deu_(i-1) )
1737:       call ABXdelta( Fsto_deu_in(i-1) * Xiss_bot_d ,  0.0d0 ,
1738:    &      Dsto_deu_, i, delta_t, 'Dsto_deu_', iabxctl)
1739:     else
1740:       F(44) = ABX ( Fsto_deu_in(i-1) * Xiss_bot_d ,
1741:    &      - Fsto_deu_out(i-1) / Nsto_deu_(i-1) , Dsto_deu_(i-1) )
1742:       call ABXdelta( Fsto_deu_in(i-1) * Xiss_bot_d ,
1743:    &      - Fsto_deu_out(i-1) / Nsto_deu_(i-1)  , Dsto_deu_, i,
1744:    &      delta_t, 'Dsto_deu_', iabxctl)
1745:     endif
1746:*
1747:* --------- Protium rich storage
1748:*
1749:
1750:     Fsto_pro_in(i) = ( 1.0 - linj_p ) * Fiss_top_(i)
1751:     if( Fsto_pro_in(i) .lt. 0.0d0 ) call negfix( Fsto_pro_in(i),
1752:    &      'Fsto_pro_in(i)', 0.0d0 )
1753:
1754:     Fsto_pro_out(i) = Fsto_out_p(i-1)
1755:     if( Fsto_pro_out(i) .lt. 0.0d0 ) call negfix( Fsto_pro_out(i),
1756:    &      'Fsto_pro_out(i)', 0.0d0 )
1757:* overall balance:
1758:
1759:     F(45) = ABX ( Fsto_pro_in(i-1) - Fsto_pro_out(i-1) , 0.0d0 ,
1760:    &      Nsto_pro_(i-1) )
1761:     call ABXdelta( Fsto_pro_in(i-1) - Fsto_pro_out(i-1) , 0.0d0  ,
1762:    &      Nsto_pro_, i, delta_t, 'Nsto_pro_', iabxctl)
1763:* total tritium:
1764:c=====================================
1765:     if( Nsto_pro_(i-1) .eq. 0.0d0 ) then
1766:        write(6,'(A,1Pe12.5)') 'Nsto_pro_=',Nsto_pro_(i-1)
1767:       F(46) = ABX ( Fsto_pro_in(i-1) * Xiss_top_t , 0.0d0
1768:    &       , Tsto_pro_(i-1) )
1769:       call ABXdelta( Fsto_pro_in(i-1) * Xiss_top_t , 0.0d0
1770:    &       , Tsto_pro_, i,  delta_t, 'Tsto_pro_', iabxctl)
1771:     else
1772:       F(46) = ABX ( Fsto_pro_in(i-1) * Xiss_top_t ,
1773:    &      - Fsto_pro_out(i-1) / Nsto_pro_(i-1) , Tsto_pro_(i-1) )
1774:       call ABXdelta( Fsto_pro_in(i-1) * Xiss_top_t ,
1775:    &      - Fsto_pro_out(i-1) / Nsto_pro_(i-1)  , Tsto_pro_, i,
1776:    &      delta_t, 'Tsto_pro_', iabxctl)
1777:     endif
1778:* deuterium balance:
1779:
1780:     if( Nsto_pro_(i-1) .eq. 0.0d0 ) then
1781:       F(47) = ABX ( Fsto_pro_in(i-1) * Xiss_top_d , 0.0d0 ,
1782:    &      Dsto_pro_(i-1) )
1783:       call ABXdelta( Fsto_pro_in(i-1) * Xiss_top_d , 0.0d0 ,
1784:    &       Dsto_pro_, i, delta_t, 'Dsto_pro_', iabxctl )
1785:     else
1786:       F(47) = ABX ( Fsto_pro_in(i-1) * Xiss_top_d ,
1787:    &      - Fsto_pro_out(i-1) / Nsto_pro_(i-1) , Dsto_pro_(i-1) )
1788:       call ABXdelta( Fsto_pro_in(i-1) * Xiss_top_d ,
1789:    &      - Fsto_pro_out(i-1) / Nsto_pro_(i-1)  , Dsto_pro_, i,
1790:    &      delta_t, 'Dsto_pro_', iabxctl )
1791:     endif
```

```
1792:*  ... Ksto_tri_ : accumulated inventory change in storage
1793:
1794:     Ksto_tri_(i) = Ksto_tri_(i-1) + ( Tsto_tri_(i-1) - Tsto_tri_(i) )
1795:
1796:     Ksto_deu_(i) = Ksto_deu_(i-1) + ( Tsto_deu_(i-1) - Tsto_deu_(i) )
1797:
1798:     Ksto_pro_(i) = Ksto_pro_(i-1) + ( Tsto_pro_(i-1) - Tsto_pro_(i) )
1799:*
1800:*-----------------------------------------------------------
1801:* Reset inventories in storage when power level changes
1802:*-----------------------------------------------------------
1803:*
1804:
1805:      ttstep = 20d0 / 3600d0
1806:*
1807:*  ... add T/D/P on every 20 seconds ...
1808:*
1809:     if ( Initial_step .eq. 0 .and. abs ( tcycle / ttstep - nint (
1810:    &       tcycle / ttstep ) ) .lt. 0.00001 ) then
1811:
1812:c      write(6,*) '======= power level changed'
1813:
1814:       Tsto_tri_(i) = Tsto_tri_(i) + Burn_t(i) * ttstep / fbu
1815:
1816:       Nsto_tri_(i) = Nsto_tri_(i) + Burn_t(i) * ttstep / fbu
1817:
1818:       Dsto_deu_(i) = Dsto_deu_(i) + Burn_d(i) * ttstep / fbu
1819:
1820:       Nsto_deu_(i) = Nsto_deu_(i) + Burn_d(i) * ttstep / fbu
1821:
1822:       Psto_pro_(i) = Psto_pro_(i) + Reac_p(i) * ttstep / fbu
1823:
1824:       Nsto_pro_(i) = Nsto_pro_(i) + Reac_p(i) * ttstep / fbu
1825:
1826:c      Tsto_tri_(i) = Tsto_tri_(i) + Burn_t(i) * ttstep
1827:
1828:c      Nsto_tri_(i) = Nsto_tri_(i) + Burn_t(i) * ttstep
1829:
1830:c      Dsto_deu_(i) = Dsto_deu_(i) + Burn_d(i) * ttstep
1831:
1832:c      Nsto_deu_(i) = Nsto_deu_(i) + Burn_d(i) * ttstep
1833:
1834:c      Psto_pro_(i) = Psto_pro_(i) + Reac_p(i) * ttstep
1835:
1836:c      Nsto_pro_(i) = Nsto_pro_(i) + Reac_p(i) * ttstep
1837:     endif
1838:c*
1839:c* Mole fraction in tritium rich storage
1840:c*
1841:
1842:     if ( Nsto_tri_(i) .ne. 0.0d0 ) then
1843:
1844:       Xsto_tri_t(i) = Tsto_tri_(i) / Nsto_tri_(i)
1845:
1846:       Xsto_tri_d(i) = Dsto_tri_(i) / Nsto_tri_(i)
1847:
1848:       Xsto_tri_p(i) = 1.0 - Xsto_tri_t(i) - Xsto_tri_d(i)
1849:
1850:     else
1851:        write(6,'(A,1Pe12.5)') 'Nsto_tri_=',Nsto_tri_(i)
1852:
1853:       Xsto_tri_t(i) = 0.0d0
1854:
1855:       Xsto_tri_d(i) = 0.0d0
1856:
1857:       Xsto_tri_p(i) = 0.0d0
1858:
1859:     endif
1860:
1861:     Psto_tri_(i) = Xsto_tri_p(i) * Nsto_tri_(i)
1862:* mobilizable tritium:
1863:
1864:     Tsto_tri_m(i) = mu_sto * Tsto_tri_(i)
1865:*
1866:* Mole fraction in deuterium rich storage
1867:*
1868:
1869:     if ( Nsto_deu_(i) .ne. 0.0d0 ) then
1870:
1871:       Xsto_deu_t(i) = Tsto_deu_(i) / Nsto_deu_(i)
1872:
1873:       Xsto_deu_d(i) = Dsto_deu_(i) / Nsto_deu_(i)
1874:
1875:       Xsto_deu_p(i) = 1.0 - Xsto_deu_t(i) - Xsto_deu_d(i)
1876:
1877:     else
1878:        write(6,'(A,1Pe12.5)') 'Nsto_deu_=',Nsto_deu_(i)
1879:
1880:       Xsto_deu_t(i) = 0.0d0
1881:
1882:       Xsto_deu_d(i) = 0.0d0
1883:
1884:       Xsto_deu_p(i) = 0.0d0
1885:
1886:     endif
1887:
1888:     Psto_deu_(i) = Xsto_deu_p(i) * Nsto_deu_(i)
1889:* mobilizable tritium:
1890:
1891:     Tsto_deu_m(i) = mu_sto * Tsto_deu_(i)
1892:* Mole fraction in deuterium rich storage
1893:
1894:     if ( Nsto_pro_(i) .ne. 0.0d0 ) then
1895:
1896:       Xsto_pro_t(i) = Tsto_pro_(i) / Nsto_pro_(i)
1897:
1898:       Xsto_pro_d(i) = Dsto_pro_(i) / Nsto_pro_(i)
1899:
1900:       Xsto_pro_p(i) = 1.0 - Xsto_pro_t(i) - Xsto_pro_d(i)
1901:
1902:     else
1903:        write(6,'(A,1Pe12.5)') 'Nsto_pro_=',Nsto_pro_(i)
1904:
1905:       Xsto_pro_t(i) = 0.0d0
```

```
1906:
1907:       Xsto_pro_d(i) = 0.0d0
1908:
1909:       Xsto_pro_p(i) = 0.0d0
1910:
1911:     endif
1912:
1913:     Psto_pro_(i) = Xsto_pro_p(i) * Nsto_pro_(i)
1914:* mobilizable tritium:
1915:
1916:     Tsto_pro_m(i) = mu_sto * Tsto_pro_(i)
1917:*
1918:*  --- Tritium/Deuterium/Protium requirements
1919:*
1920:
1921:     Fsto_tri_out_t(i) = Fsto_tri_out(i) * Xsto_tri_t(i)
1922:
1923:     Fsto_deu_out_d(i) = Fsto_deu_out(i) * Xsto_deu_d(i)
1924:
1925:     Fsto_pro_out_p(i) = Fsto_out_p(i) * Xsto_pro_p(i)
1926:*
1927:*  ... Fueling ..........................
1928:*
1929:
1930:     Fuelf_t(i) = Fuel_t(i) - Fplas_nb_t(i-1) - Fplas_prop_t(i-1)
1931:     if( Fuelf_t(i) .lt. 0.0d0 ) call negfix( Fuelf_t(i),
1932:    &       'Fuelf_t(i)', 0.0d0 )
1933:
1934:
1935:     return
1936:     end
1937:C==================================================
1938:C== Funtions ==================================
1939:C==================================================
1940:     subroutine negfix( X , name, fix )
1941:C -------------------------------------------------------
1942:C  negative value fixup with a message
1943:C -------------------------------------------------------
1944:     double precision X,fix
1945:     character*(*) name
1946:     if( X.lt.0.0d0 ) then
1947:c       write(6,'(2A,2(A,1PE15.8))') '!!! negative value of ',
1948:c   &     name,' (=',X,') is fixed to ',fix
1949:       X = fix
1950:     end if
1951:     return
1952:     end
1953:c
1954:     subroutine ABXdelta( A0,B0,X,i,delta_t, xname, iabxctl )
1955:C -------------------------------------------------------
1956:C  time promotion from X(i-1) to X(i) when
1957:C   dX/dt = A + B*X
1958:C -------------------------------------------------------
1959:     implicit double precision(a-h,o-z)
1960:c
1961:     double precision a,b,delta_t,A0,B0
1962:     integer i
1963:     double precision X(0:*)
1964:     character*(*) xname
1965:C
1966:     factor = 3600.d0
1967:     if( iabxctl .eq. 0 )  then
1968:        A= A0/FACTOR
1969:        B= B0/FACTOR
1970:     else
1971:        A= A0
1972:        B= B0
1973:     endif
1974:C
1975:C -------------------------------------------------------
1976:C   analytic form
1977:C -------------------------------------------------------
1978:C
1979:*       if( abs(B) .gt. 1.0d-10*abs(A) ) then
1980:*         AB = A/B
1981:*         X(i) =  ( X(i-1) + AB ) * exp( B*delta_t ) - AB
1982:*       else
1983:*         X(i) = X(i-1) + (A + B*X(i-1))*delta_t
1984:*       endif
1985:C
1986:C -------------------------------------------------------
1987:C   explicit Euler
1988:C -------------------------------------------------------
1989:C
1990:*       X(i) =  X(i-1) + ( A + B*X(i-1) ) * delta_t
1991:C
1992:C -------------------------------------------------------
1993:C   implicit Euler
1994:C -------------------------------------------------------
1995:Ccccc   X(i) =  X(i-1) + ( A + B*X(i) ) * delta_t
1996:C
1997:*       BB = 1.0d0 - B*delta_t
1998:*       if( abs(BB) .gt. 1.0d-30 ) then
1999:*         X(i) =  (X(i-1) + A* delta_t) / BB
2000:*       else
2001:*         X(i) =  X(i-1) + ( A + B*X(i-1) ) * delta_t
2002:*       endif
2003:C
2004:C -------------------------------------------------------
2005:C   like 4'th order Runge-Kutta
2006:C -------------------------------------------------------
2007:C
2008:        R1  =  delta_t * ( A + B *  X(i-1) )
2009:        R2  =  delta_t * ( A + B * ( X(i-1) + 0.5d0*R1 ) )
2010:        R3  =  delta_t * ( A + B * ( X(i-1) + 0.5d0*R2 ) )
2011:        R4  =  delta_t * ( A + B * ( X(i-1) + R3 ) )
2012:        X(i) =  X(i-1) + (1d0/6d0) * ( R1 + 2*R2 + 2*R3 + R4 )
2013:C
2014:     return
2015:     end
2016:*****************************************************************
2017:* function FPPOWER : Fusion power profile ( 0 < fppower <= 1 )
2018:*
2019:*  Rump up & rumpdown are given in steps of 5 sec width.
```

```
2020:*********************************************************************
2021:     double precision function FPPOWER(TCYCLE,T_BURN,T_P_RU,T_P_RD
2022:    &                          , T_DWELL )
2023:
2024:     double precision TCYCLE, T_BURN, T_P_RU, T_P_RD, T_DWELL
2025:     double precision TSTEP,DNU,DND
2026:
2027:c    parameter ( TSTEP = 5d0/3600d0 )
2028:     TSTEP = T_P_RU/10d0
2029:c    DNU = nint( T_P_RU / TSTEP )
2030:     DND = nint( T_P_RD / TSTEP )
2031:     DNU = 10.0d0
2032:
2033:     if ( TCYCLE.lt.T_P_RU*0.999 ) then
2034:       FPPOWER =  int((TCYCLE+1.0d-9)/TSTEP) / DNU
2035:     else if ( TCYCLE.lt.(T_P_RU+T_BURN)*0.999 ) then
2036:       FPPOWER = 1.0D0
2037:     else if ( TCYCLE.lt.(T_P_RU+T_BURN+T_P_RD)*0.999 ) then
2038:       FPPOWER = int((T_P_RU+T_BURN+T_P_RD-TCYCLE-1.0d-9)
2039:    &            /TSTEP)/DND
2040:     else
2041:       FPPOWER = 0.0D0
2042:     end if
2043:     return
2044:     end
2045:*********************************************************************
2046:*  function FPWIDTH
2047:*   Step width when rump up & rumpdown are given in steps
2048:*********************************************************************
2049:     double precision function FPWIDTH(TCYCLE,T_BURN,T_P_RU,T_P_RD)
2050:
2051:     double precision TCYCLE, T_BURN, T_P_RU, T_P_RD
2052:     double precision TSTEP
2053:     parameter ( TSTEP = 20d0/3600d0 )
2054:
2055:     if ( TCYCLE.lt.T_P_RU*0.999 ) then
2056:       FPWIDTH = TSTEP
2057:     else if ( TCYCLE.lt.(T_P_RU+T_BURN)*0.999 ) then
2058:       FPWIDTH = T_BURN
2059:     else
2060:       FPWIDTH = TSTEP
2061:     end if
2062:     return
2063:     end
2064:*********************************************************************
2065:* function FPPOWER0
2066:*  rump up & rumdown are continuous.
2067:*********************************************************************
2068:     double precision function FPPOWER0(TCYCLE,T_BURN,T_P_RU,T_P_RD
2069:    &                          , T_DWELL )
2070:
2071:     double precision TCYCLE, T_BURN, T_P_RU, T_P_RD, T_DWELL
2072:
2073:     if ( TCYCLE.lt.T_P_RU*0.999 ) then
2074:       FPPOWER0 = TCYCLE/T_P_RU
2075:     else if ( TCYCLE.lt.(T_P_RU+T_BURN)*0.999 ) then
2076:       FPPOWER0 = 1.0D0
2077:     else if ( TCYCLE.lt.(T_P_RU+T_BURN+T_P_RD)*0.999 ) then
2078:       FPPOWER0 = (1.0D0 - (TCYCLE-(T_P_RU+T_BURN))/T_P_RD)
2079:     else
2080:       FPPOWER0 = 0.0D0
2081:     end if
2082:     return
2083:     end
2084:*********************************************************************
2085:*  function fnbpower : neutral beam injection profile.
2086:*********************************************************************
2087:     double precision function FNBPOWER(TCYCLE,T_BURN,T_P_RU,T_P_RD,
2088:    &       T_NB_RU,T_NB_RD)
2089:
2090:     double precision TCYCLE, T_BURN, T_P_RU, T_P_RD, T_NB_RU, T_NB_RD
2091:
2092:     if ( TCYCLE.lt.(T_P_RU-T_NB_RU) ) then
2093:       FNBPOWER   = 0.0D0
2094:     else if ( TCYCLE.lt.T_P_RU ) then
2095:       FNBPOWER   = (TCYCLE-(T_P_RU-T_NB_RU)) /(T_P_RU-T_NB_RU)
2096:     else if ( TCYCLE.lt.(T_P_RU+T_BURN) ) then
2097:       FNBPOWER   = 1.0D0
2098:     else if ( TCYCLE.lt.(T_P_RU+T_BURN+T_NB_RD) ) then
2099:       FNBPOWER   = 1.0D0 - (TCYCLE-(T_P_RU+T_BURN))/(T_P_RD-T_NB_RD)
2100:     else
2101:       FNBPOWER   = 0.0D0
2102:     end if
2103:     return
2104:     end
2105:*********************************************************************
2106:*  function Sigma_tend : return sum of tend(j)
2107:*********************************************************************
2108:     double precision function SIGMA_TEND(T, tend)
2109:
2110:     double precision T, tend
2111:C*
2112:C*  ... Sigma_tend(j) is always zero for 1997 case ( t <1000 sec)
2113:C*
2114:     SIGMA_TEND  = tend
2115:     return
2116:     end
2117:*********************************************************************
2118:*    function f(M) : 0.1 if Be is used, otherwise 1.0
2119:*********************************************************************
2120:     double precision function FM(M)
2121:
2122:     double precision M
2123:c    integer NINF
2124:
2125:ccccc if ( M.eq.'Be' ) then
2126:     if ( M.eq.2 ) then
2127:       FM     = 0.1D0
2128:     else
2129:       FM     = 1.0d0
2130:     end if
2131:     return
2132:     end
2133:*********************************************************************
```

page 63

```
2134:*    function rho(M) : material density (kg/m3)
2135:*********************************************************************
2136:     double precision function RHO(M)
2137:
2138:     double precision M
2139:     integer NINRHO
2140:
2141:     NINRHO  = 1
2142:ccccc if ( M.eq.'C' ) then
2143:     if ( M.eq.1 ) then
2144:       RHO    = 2.3D3
2145:ccccc else if ( M.eq.'Be' ) then
2146:     else if ( M.eq.2 ) then
2147:       RHO    = 1.8D3
2148:ccccc else if ( M.eq.'W' ) then
2149:     else if ( M.eq.3 ) then
2150:       RHO    = 19.3D3
2151:     else
2152:       NINRHO  = 0
2153:     end if
2154:
2155:     if ( NINRHO.eq.0 ) then
2156:       write(6,*) 'xxx(function rho) no matching meterial ', ' <', M,
2157:    &         '>'
2158:       stop 888
2159:     end if
2160:     return
2161:     end
2162:*********************************************************************
2163:*    function Rc(M,T) : codeposition rate (kg tritium/burn*year)
2164:*********************************************************************
2165:     double precision function RC(MEDGE,TEDGE)
2166:
2167:     double precision MEDGE, TEDGE
2168:
2169:     NINRC   = 1
2170:ccccc if(Medge.eq.'C') then
2171:     if ( MEDGE.eq.1 ) then
2172:       if ( TEDGE.eq.150 ) then
2173:         RC     = 155
2174:       else if ( TEDGE.eq.100 ) then
2175:         RC     = 135
2176:       else if ( TEDGE.eq.60 ) then
2177:         RC     = 88
2178:       else if ( TEDGE.eq.130 ) then
2179:         RC     = 130
2180:       else
2181:         NINRC   = 0
2182:       end if
2183:cccccc  else if(Medge.eq.'Be') then
2184:     else if ( MEDGE.eq.2 ) then
2185:       if ( TEDGE.eq.150 ) then
2186:         RC     = 40
2187:       else if ( TEDGE.eq.60 ) then
2188:         RC     = 33
2189:       else if ( TEDGE.eq.30 ) then
2190:         RC     = 18
2191:       else
2192:         NINRC   = 0
2193:       end if
2194:cccc   else if(Medge.eq.'W') then
2195:     else if ( MEDGE.eq.3 ) then
2196:       if ( TEDGE.eq.40 ) then
2197:         RC     = 0.005
2198:       else
2199:         NINRC   = 0
2200:       end if
2201:     else
2202:       NINRC   = 0
2203:     end if
2204:
2205:     if ( NINRC.eq.0 ) then
2206:       write(6,*) 'xxx(function Rc) no matching meterial or',
2207:    &         ' temperature <', MEDGE, '> T= ', TEDGE
2208:       stop 888
2209:     end if
2210:     return
2211:     end
2212:*********************************************************************
2213:*    function Ed(M,T)
2214:*********************************************************************
2215:     double precision function ED(MEDGE,TEDGE)
2216:
2217:     double precision MEDGE, TEDGE
2218:     NINED   = 1
2219:
2220:ccccc if(Medge.eq.'C') then
2221:     if ( MEDGE.eq.1 ) then
2222:       if ( TEDGE.eq.150D0 ) then
2223:         ED     = 0.27D0
2224:       else if ( TEDGE.eq.100D0 ) then
2225:         ED     = 0.28D0
2226:       else if ( TEDGE.eq.60D0 ) then
2227:         ED     = 0.23D0
2228:       else if ( TEDGE.eq.30D0 ) then
2229:         ED     = 0.17D0
2230:       else
2231:         NINED   = 0
2232:       end if
2233:ccccc else if(Medge.eq.'Be') then
2234:     else if ( MEDGE.eq.2 ) then
2235:       if ( TEDGE.eq.150D0 ) then
2236:         ED     = 0.46D0
2237:       else if ( TEDGE.eq.60D0 ) then
2238:         ED     = 0.73D0
2239:       else if ( TEDGE.eq.30D0 ) then
2240:         ED     = 0.49D0
2241:       else
2242:         NINED   = 0
2243:       end if
2244:ccccc else if(Medge.eq.'W') then
2245:     else if ( MEDGE.eq.3 ) then
2246:       if ( TEDGE.eq.40D0 ) then
2247:         ED     = 0.007D0
```

```
2248:        else
2249:          NINED  = 0
2250:        end if
2251:      else
2252:        NINED  = 0
2253:      end if
2254:
2255:      if ( NINED.eq.0 ) then
2256:        write(6,*) 'xxx(function Ed) no matching meterial or',
2257:     &          ' temperature: M = <', MEDGE, '> T= ', TEDGE
2258:        stop 888
2259:      end if
2260:      return
2261:      end
2262:
2263:*************************************************************************
2264:*    function I(T,t)
2265:*
2266:* Tritium inventry from diverter by log-linear interpolation.
2267:*************************************************************************
2268:      double precision function AI(TEMP,T)
2269:
2270:      double precision TEMP, T
2271:      parameter( NUMT = 6 )
2272:      parameter( NTEMP   = 3 )
2273:      double precision TI(NUMT)
2274:      double precision IV(NUMT,NTEMP)
2275:      double precision TEMPS(NTEMP)
2276:c ------------------------------------------------------------------
2277:      data TI /278D0, 417D0, 694D0, 1.39D3, 2.78D3, 4.86D3/
2278:
2279:      data TEMPS(1) /1000.D0/
2280:      data(IV(I,1),I=1,NUMT) /14D0, 260D0, 400D0, 420D0, 440D0, 460D0/
2281:      data TEMPS(2) /1400.D0/
2282:      data(IV(I,2),I=1,NUMT) /14D0, 260D0, 360D0, 380D0, 400D0, 410D0/
2283:      data TEMPS(3) /1800.D0/
2284:      data(IV(I,3),I=1,NUMT) /14D0, 260D0, 360D0, 380D0, 400D0, 410D0/
2285:c ------------------------------------------------------------------
2286:
2287:      if ( TEMP.eq.TEMPS(1) ) then
2288:        K     = 1
2289:      else if ( TEMP.eq.TEMPS(2) ) then
2290:        K     = 2
2291:      else if ( TEMP.eq.TEMPS(3) ) then
2292:        K     = 3
2293:      else
2294:        write(6,*) 'xxx(function ai) no matching temperature ', TEMP
2295:        stop 888
2296:      end if
2297:
2298:      do II = 1, NUMT
2299:        if ( TI(II).ge.T ) go to 100
2300:      end do
2301:      II     = NUMT - 1
2302:  100 continue
2303:C
2304:c=========================================
2305:      if( (TI(II+1)-TI(II) ) .eq. 0.0d0 ) then
2306:c       write(6,*) 'AI ',(TI(II+1)-TI(II))
2307:        AI = EXP( LOG(IV(II,K)) )
2308:      else
2309:        AI = EXP((LOG(IV(II+1,K))-LOG(IV(II,K)))/(TI(II+1)-TI(II))*
2310:     &      (T-TI(II))+LOG(IV(II,K)))
2311:      endif
2312:C
2313:      return
2314:      end
2315:c
2316:c
2317:      double precision function condvar( i, j, A, PP )
2318:C ------------------------------------------------------------------
2319:C  conddition value of A
2320:C ------------------------------------------------------------------
2321:      implicit double precision(a-h,o-z)
2322:c
2323:      double precision A(j,j), B(3,3)
2324:C                  i : used size   j : dimension size
2325:c     norm (A)
2326:      ANORM = 0.d0
2327:      do ii = 1,i
2328:        do jj = 1,i
2329:          if( abs(A(ii,jj)) .gt. ANORM )  ANORM = abs(A(ii,jj))
2330:        end do
2331:      end do
2332:c
2333:c     norm (1/A)
2334:      if( i .eq. 2 )  then
2335:        B(1,1) = A(2,2)
2336:        B(2,2) = A(1,1)
2337:        B(2,1) = -A(2,1)
2338:        B(1,2) = -A(1,2)
2339:      else
2340:        B(1,1) = A(2,2)/(A(1,1)*A(2,2)-A(1,2)*A(2,1))
2341:        B(1,2) = A(1,2)/(A(1,2)*A(2,1)-A(1,1)*A(2,2))
2342:        B(1,3) = A(2,2)/(A(2,2)*A(3,1)-A(2,1)*A(3,2))
2343:        B(2,1) = A(2,1)/(A(1,2)*A(2,1)-A(1,1)*A(2,2))
2344:        B(2,2) = A(1,1)/(A(1,1)*A(2,2)-A(1,2)*A(2,1))
2345:        B(2,3) = A(2,1)/(A(2,1)*A(3,2)-A(2,2)*A(3,1))
2346:        B(3,1) = 0.d0
2347:        B(3,2) = 0.d0
2348:        B(3,3) = 0.d0
2349:      end if
2350:      CANORM = 0.d0
2351:      do ii = 1,i
2352:        do jj = 1,i
2353:          if( i .eq. 2 ) B(ii,jj) = B(ii,jj) / PP
2354:          if( abs(B(ii,jj)) .gt. CANORM )  CANORM = abs(B(ii,jj))
2355:        end do
2356:      end do
2357:      condvar = ANORM * CANORM
2358:c
2359:      return
2360:      end
```

page 65

● _output.f

```
1:      subroutine output ( ic, nstartp )
 2:c=====================================================================
 3:c      Fusion reactor tritium inventory evaluation system
 4:c---------------------------------------------------------------------
 5:c  output calculated results
 6:c---------------------------------------------------------------------
 7:c $Id: _output.f,v 1.11 1998/03/26 05:22:22 sasaki Exp $
 8:c=====================================================================
 9:c
10:c                          ic : cycle no.
11:      integer     ic
12:cccc  implicit real*8 (a-h,o-z)
13:c
14:      include '_sizes.h'
15:      include '_commons1.h'
16:      include 'varconst.h'
17:c
18:      logical opd
19:c ------------------------------------------------------------
20:c
21:c  .... one file per variable  output
22:c
23:      write(6,*) '======= Output results of simulation on files ===',
24:     &        ' nvarout= ', nvarout
25:c
26:      iunit =10
27:      inquire( unit=iunit, opened=opd )
28:      if( opd ) close (iunit)
29:c
30:      if(OUTPUTALL.eq.0) then
31:c
32:        do i=1,nvarout
33:          call fileoutput( ic, varout(i), ivarout(i), iunit, ierr,
34:     &                     nstartp )
35:        end do
36:c
37:      else
38:c
39:        do i=1,nvar
40:          kk = index(varname(i), '(t)')
41:          if( kk.ne.0 ) then
42:            kk =kk - 1
43:          else
44:            kk = len(varname(i))
45:          endif
46:          call fileoutput( ic, varname(i)(:kk), i, iunit, ierr,
47:     &                     nstartp )
48:        end do
49:c
50:      end if
51:c
52:      return
53:      end
54:c
55:c
56:c
57:      subroutine fileoutput( ic, outvar, ivaro, iunit, ierr, nstartp )
58:c=====================================================================
59:c   output results of variable named outvar and sequentian # ivaro
60:c   on I/O unit iunit.
61:c=====================================================================
62:      character*(*) outvar
63:      integer     ic, ivaro, nstartp
64:c
65:      include '_sizes.h'
66:      include '_commons1.h'
67:      include 'varconst.h'
68:c
69:      character*32 unit
70:c
71:      do i=len(outvar),1,-1
72:        if( outvar(i:i).ne.' ' ) then
73:          lv = i
74:          goto 110
75:        end if
76:      end do
77:      lv = len(outvar)
78:  110 continue
79:c
80:      call fileopen( ic, result(:llr), outvar(:lv), iunit, ierr )
81:      if(ierr.ne.0) then
82:        write(6,*) 'XXX failed to open output file ',outvar(:lv)
83:        if( sysname.eq.'Macintosh' ) pause 'PAUSE'
84:        stop 888
85:      end if
86:c
87:      unit = ' '
88:      if( outvar(1:1).eq.'F' .or.
89:     &   outvar(1:1).eq.'R' ) then
90:        unit = 'Mole/hour'
91:      else if( outvar(1:1).eq.'T' .or.
92:     &       outvar(1:1).eq.'D' .or.
93:     &       outvar(1:1).eq.'P' .or.
94:     &       outvar(1:1).eq.'N' .or.
95:     &       outvar(1:1).eq.'C' .or.
96:     &       outvar(1:1).eq.'H' .or.
97:     &       outvar(1:1).eq.'A' ) then
98:        unit = 'Moles'
99:      else if( outvar(1:4).eq.'Burn' ) then
100:       unit = 'Mega Watts'
101:     else if( outvar(1:5).eq.'res_N' ) then
102:       unit = 'cm-3'
103:     else
104:       unit = 'arbitrary unit'
105:     end if
106:c
107:     do i=len(unit),1,-1
108:       if( unit(i:i).ne.' ' ) then
109:         lu = i
110:         goto 130
111:       end if
```

```
112:     end do
113:     lu = len(unit)
114:  130 continue
115:
116:c
117:c   ... char(9) is ASC II Tab character ...
118:c
119:     if((mode(1:3) .eq. 'RUN' .or. mode(1:3) .eq. 'RES') .and.
120:    &   ic .le. nstartc )                          then
121:       if(sysname.eq.'UNIX') then
122:         write(iunit,'(5a)') '#Time(sec)',char(9),outvar(:lv),
123:    &          char(9),unit(:lu)
124:       else
125:cc        write(iunit,'(5a)') 'Time(sec)',char(9),outvar(:lv),
126:cc  &          char(9),unit(:lu)
127:         write(iunit,'(5a)') 'Time(sec)',char(9),outvar(:lv)
128:       endif
129:       if( ivaro .eq. 999 ) then
130:         write(iunit,7100) (tmax*(ic-1)+times_r(1))*3600,
131:    &                  char(9),res_N(1)
132:       else
133:         if( ivaro .gt. 900 .and. ivaro .le. 900+NT_PUMP ) then
134:           if( mode(1:3) .eq. 'RUN' .and. nstartc .eq. 1 ) then
135:             write(iunit,7100) (tmax*(ic-1)+times(0))*3600,
136:    &                  char(9),Tpv_i(0,ivaro-900)
137:           else if( mode(1:3) .eq. 'RES' ) then
138:             write(iunit,7100) (tmax*(ic-1)+times_r(1))*3600,
139:    &                  char(9),Tpv_i(1,ivaro-900)
140:           endif
141:         else
142:           if( mode(1:3) .eq. 'RUN' .and. nstartc .eq. 1 ) then
143:             write(iunit,7100) (tmax*(ic-1)+times(0))*3600,
144:    &                  char(9),XVARS(0,ivaro)
145:           else if( mode(1:3) .eq. 'RES' ) then
146:             write(iunit,7100) (tmax*(ic-1)+times_r(1))*3600,
147:    &                  char(9),XVARS(1,ivaro)
148:           endif
149:         endif
150:       endif
151:     endif
152:     if( ivaro .eq. 999 ) then
153:       if( mode(1:3) .eq. 'RRR' ) then
154:         do it=nstartp,ntimes
155:           write(iunit,7100) (tmax*(ic-1)+times(it))*3600
156:    &                ,char(9),res_N(it)
157:         end do
158:       else
159:         do it=2,ntimes_r
160:           write(iunit,7100) (tmax*(ic-1)+times_r(it))*3600
161:    &                ,char(9),res_N(it)
162:         end do
163:       end if
164:     else
165:       if( ivaro .gt. 900 .and. ivaro .le. 900+NT_PUMP ) then
166:         if( mode(1:3) .eq. 'RUN' .or. mode(1:3) .eq. 'RRR' ) then
167:           do it=nstartp,ntimes
168:             write(iunit,7100) (tmax*(ic-1)+times(it))*3600
169:    &                ,char(9),Tpv_i(it,ivaro-900)
170:           end do
171:         else
172:           do it=2,ntimes_r
173:             write(iunit,7100) (tmax*(ic-1)+times_r(it))*3600
174:    &                ,char(9),Tpv_i(it,ivaro-900)
175:           end do
176:         end if
177:       else
178:         if( mode(1:3) .eq. 'RUN' .or. mode(1:3) .eq. 'RRR' ) then
179:           do it=nstartp,ntimes
180:             write(iunit,7100) (tmax*(ic-1)+times(it))*3600
181:    &                ,char(9),XVARS(it,ivaro)
182:           end do
183:         else
184:           do it=2,ntimes_r
185:             write(iunit,7100) (tmax*(ic-1)+times_r(it))*3600
186:    &                ,char(9),XVARS(it,ivaro)
187:           end do
188:         end if
189:       end if
190:     endif
191: 7100 format(1pe12.5,a,1x,e12.5)
192:c
193:     close(iunit)
194:     return
195:     end
196:c ==================================================================
197:c  fileopen routine opens file under specified directory/folder
198:c ==================================================================
199:     subroutine fileopen( ic, path, fname, iunit, ierr )
200:c------------------------------------------------------------------
201:c  Open a file named as "PATH + <derectory separator> + FNAME" on
202:c  I/O unit IUNIT.
203:c
204:c   Return open error code on IERR
205:c------------------------------------------------------------------
206:     integer      ic
207:     character*(*) fname, path
208:c
209:     include '_sizes.h'
210:     include '_commons1.h'
211:c
212:     character*256 ff
213:c
214:     ierr = 0
215:c
216:     ff = ' '
217:c
218:     if( path .eq. ' ' ) then
219:       ff = fname
220:     else
221:c      lp = index(path,' ') - 1
222:c      if(lp.lt.0) lp = len(path)
223:c
224:c   ... add directory separator ...
225:c
```

page 67

```
226:c  ":" for Macintosh, "/" for UNIX, "¥" or char(92) for MS-DOS/Windows
227:c
228:       if( sysname.eq.'Macintosh'.or.index(sysname,'Mac').ne.0
229:    &     .or. index(sysname,'MAC').ne.0 ) then
230:
231:         ff = path(:len(path))//':'//fname
232:
233:       else if( sysname.eq.'UNIX' .or. index(sysname,'UNIX').ne.0)
234:    &      then
235:
236:         ff = path(:len(path))//'/'//fname
237:
238:       else if( sysname.eq.'Windows'
239:    &      .or. index(sysname,'Win').ne.0
240:    &      .or. index(sysname,'WIN').ne.0
241:    &      .or. index(sysname,'DOS').ne.0
242:    &      ) then
243:
244:         ff = path(:len(path))//char(92)//fname
245:
246:       else
247:
248:         write(6,*) 'XXX cannot compose file pathname because of',
249:    &      ' unknown system <',sysname,'> is specified.'
250:         stop 888
251:
252:       end if
253:     end if
254:c
255:c    lf = index(ff,' ') - 1
256:c    if(lf.lt.0) lf = len(ff)
257:     do lf=len(ff),1,-1
258:       if( ff(lf:lf).ne.' ' ) goto 200
259:     end do
260:     lf = len(ff)
261:  200 continue
262:c
263:c***     write(6,*) '--- opening a file <',ff(:lf),'> on I/O unit ',
264:c***    &          iunit
265:c
266:c***     if(sysname.eq.'Macintosh') then
267:c***       write(*,*) '--- opening a file <',ff(:lf),'> on I/O unit ',
268:c***    &          iunit
269:c***     end if
270:c
271:c
272:     if((mode(1:3) .eq. 'RUN' .or. mode(1:3) .eq. 'RES') .and.
273:    &  ic .le. nstartc )                                  then
274:     open( iunit, file=ff(:lf), status='UNKNOWN', form='FORMATTED',
275:    &     iostat = ios )
276:     else
277:     open( iunit, file=ff(:lf), status='UNKNOWN', form='FORMATTED',
278:    &     iostat = ios , access = 'APPEND' )
279:     end if
280:c
281:c
282:     if( ios.ne.0 ) then
283:       write(6,*) '--- failed to open a file <',ff(:lf),'> code=',ios
284:       ierr = ios
285:
286:       if(sysname.eq.'Macintosh') then
287:         write(*,*) '--- failed to open a file <',ff(:lf),'> code=',
288:    &              ios
289:       end if
290:     end if
291:c
292:     return
293:     End
294:
295:c ================================================================
296:c response file output routine
297:c ================================================================
298:     subroutine rstoutfst( )
299:c----------------------------------------------------------------
300:c I/O unit 20.
301:c
302:c  output items
303:c        TMAX, TEND, Tmp_tend
304:c        &OUTPUT( if OUTPUTALL=0)
305:c           nvarout, varout(1:nvarout), ivarout(1:nvarout)
306:c        &CONSTANT
307:c----------------------------------------------------------------
308:     include '_sizes.h'
309:     include '_commons1.h'
310:     include 'varconst.h'
311:c
312:c.......... first
313:     kout = 20
314:     open(kout, file=response, status='UNKNOWN',
315:    &        form='FORMATTED', iostat = ios )
316:     if( ios .ne. 0 )      then
317:         write(6,*) 'XXX failed to open response file '
318:         if( sysname.eq.'Macintosh' ) pause 'PAUSE'
319:         stop 888
320:     endif
321:
322:     write(kout,'(A55)')
323:    & 'Tritium Inventory Evaluation System Step Response Input'
324:     write(kout,'(A11)') 'Version 1.0'
325:     write(kout,'(A12,1PE12.5,A5)') 'TMAX       ',
326:    &             tmax*3600d0,' sec'
327:     write(kout,'(A7,A128)') 'RESULT=',result
328:     if( OUTPUTALL .eq. 0 ) then
329:         write(kout,'(A20,I5)') '***OUTPUT VARIABLES ',nvarout
330:         do i=1,nvarout
331:           write(kout,'(I5,3X,A32)') ivarout(i),varout(i)
332:         end do
333:     else
334:         write(kout,'(A13)') '***OUTPUT ALL'
335:     endif
336:     write(kout,'(A20,I5)') '***CONSTANT VALUES  ',nconst
337:     do i=1,nconst
338:         write(kout,'(A32,1PE15.8)') constname(i), C(i)
339:     end do
```

```
340:     write(kout,'(A20,I5)') '***VARIABLES       ', NVAR
341:
342:     close ( kout )
343:c
344:     return
345:     End
346:
347:c ==================================================================
348:     subroutine rstoutput( ip )
349:c------------------------------------------------------------------
350:c  I/O unit 20.
351:c
352:c  output items
353:c        tend, Timp_tend
354:c        nbreakc(ip), nbreakp(ip)
355:c        XVARS(nbreakp(ip), 1:NVAR)
356:c        Tpv_i(nbreakp(ip), 1:NT_PUMP)
357:c        Dpv_i(nbreakp(ip), 1:NT_PUMP)
358:c        Ppv_i(nbreakp(ip), 1:NT_PUMP)
359:c        Hepv_i(nbreakp(ip), 1:NT_PUMP)
360:c        Rt_rem_i(nbreakp(ip), 1:NT_PUMP)
361:c        Rd_rem_i(nbreakp(ip), 1:NT_PUMP)
362:c        Rp_rem_i(nbreakp(ip), 1:NT_PUMP)
363:c        Rhe_rem_i(nbreakp(ip), 1:NT_PUMP)
364:c------------------------------------------------------------------
365:c
366:     include '_sizes.h'
367:     include '_commons1.h'
368:     include 'varconst.h'
369:c
370:     kout = 20
371:     open(kout, file=response,  status='UNKNOWN',
372:    &         form='FORMATTED', access='APPEND', iostat = ios )
373:     if( ios .ne. 0 )      then
374:         write(6,*) 'XXX failed to open response file '
375:         if( sysname.eq.'Macintosh' ) pause 'PAUSE'
376:         stop 888
377:     endif
378:c
379:     write(kout,'(A12)') '------------'
380:     write(kout,'(A12,I5,A6)') 'Break point ',nbreakc(ip), ' cycle'
381:     write(kout,'(A12,I5,A4)') '     time  ',
382:    &   int(dbreakp(ip)*3600d0), ' sec'
383:     write(kout,'(A12,1PE12.5,A5)') 'TEND       ',
384:    &             tend*3600d0,' sec'
385:     write(kout,'(A12,1PE12.5,A5)') 'Tmp_tend    ',
386:    &             Timp_tend,' Moles'
387:     do i= 1, NVAR
388:       write(kout,'(A32,1P2E12.5)') VARNAME(i),XVARS(0,i),
389:    &                        XVARS(nbreakp(ip),i)
390:     end do
391:     write(kout,'(A20)') '***PUMP VARIABLES   '
392:     write(kout,'(A8)')  'Tpv_i   '
393:     write(kout,'(1P24E12.5)') (Tpv_i(0,i),i=1,NT_PUMP)
394:     write(kout,'(1P24E12.5)') (Tpv_i(nbreakp(ip),i),i=1,NT_PUMP)
395:     write(kout,'(A8)')  'Dpv_i   '
396:     write(kout,'(1P24E12.5)') (Dpv_i(0,i),i=1,NT_PUMP)
397:     write(kout,'(1P24E12.5)') (Dpv_i(nbreakp(ip),i),i=1,NT_PUMP)
398:     write(kout,'(A8)')  'Ppv_i   '
399:     write(kout,'(1P24E12.5)') (Ppv_i(0,i),i=1,NT_PUMP)
400:     write(kout,'(1P24E12.5)') (Ppv_i(nbreakp(ip),i),i=1,NT_PUMP)
401:     write(kout,'(A8)')  'Hepv_i  '
402:     write(kout,'(1P24E12.5)') (Hepv_i(0,i),i=1,NT_PUMP)
403:     write(kout,'(1P24E12.5)') (Hepv_i(nbreakp(ip),i),i=1,NT_PUMP)
404:     write(kout,'(A8)')  'Rt_rem_i'
405:     write(kout,'(1P24E12.5)') (Rt_rem_i(0,i),i=1,NT_PUMP)
406:     write(kout,'(1P24E12.5)') (Rt_rem_i(nbreakp(ip),i),i=1,NT_PUMP)
407:     write(kout,'(A8)')  'Rd_rem_i'
408:     write(kout,'(1P24E12.5)') (Rd_rem_i(0,i),i=1,NT_PUMP)
409:     write(kout,'(1P24E12.5)') (Rd_rem_i(nbreakp(ip),i),i=1,NT_PUMP)
410:     write(kout,'(A8)')  'Rp_rem_i'
411:     write(kout,'(1P24E12.5)') (Rp_rem_i(0,i),i=1,NT_PUMP)
412:     write(kout,'(1P24E12.5)') (Rp_rem_i(nbreakp(ip),i),i=1,NT_PUMP)
413:     write(kout,'(A9)')  'Rhe_rem_i'
414:     write(kout,'(1P24E12.5)') (Rhe_rem_i(0,i),i=1,NT_PUMP)
415:     write(kout,'(1P24E12.5)') (Rhe_rem_i(nbreakp(ip),i),i=1,NT_PUMP)
416:c
417:     close ( kout )
418:
419:     return
420:     End
```

● _stepresp.f

```
 1:     subroutine respinit
 2:C=====================================================================
 3:C      Step response  -- S0+deltaS --  data prepare
 4:C                     made     1998.02.09
 5:C---------------------------------------------------------------------
 6:C  input problem control DATA  mode = 'RES'
 7:C  calculate S0+deltaS, res_N()  for NRESPTIME(1)
 8:C
 9:C=====================================================================
10:     implicit real*8 (A-H,O-Z)
11:C
12:     include '_sizes.h'
13:C
14:     include '_commons1.h'
15:C
16:     include 'varconst.h'
17:C
18:c...... calculate delat_t
19:     sgmT = 2*4.d-16
20:     dntauP = 1.0d14
21:     A_Burn_t_sec = Burn_t(1) * No / 3600d0
22:cccccccccccccccc  sgmT * n * n * volume = A_Burn_t_sec
23:     res_N(0) = sqrt( A_Burn_t_sec /( volume * sgmT ) )
24:     res_N(1) = res_N(0)
```

page 69

```
25:
26:     if( res_N(0) .eq. 0.0d0 )  then
27:        tauP = 0.0d0
28:     else
29:        tauP = dntauP / res_N(0)
30:     endif
31:     if( tauP .eq. 0.0d0 )  then
32:       S0 = res_N(0) * res_N(0) * sgmT
33:     else
34:       S0 = res_N(0) / tauP + res_N(0) * res_N(0) * sgmT
35:     endif
36:     write(6,'(4(A,1PE12.5))') 'tauP=',tauP,'  Burn_t=',Burn_t(1),
37:    &   '  A_Burn_t_sec=',A_Burn_t_sec,'  volume=',volume
38:c
39:     drespe = NRESPTIME(1)
40:c            DELTAS (%)
41:     DELTAS = S0 * DELTAS
42:     S1 = S0 + DELTAS
43:c
44:     times_r(0) =  0.d0
45:     times_r(1) =  nstartsec
46:
47:     write(6,'(3(A,1PE12.5))')   '   1  N=',res_N(1),
48:    &   '  S=',S1,'  times=',times_r(1)
49:c
50:     do i = 2, MAXTIMES
51:       delta_t = drespe / 50d0
52:       if( res_N(i-1) .eq. 0.0d0 )  then
53:          tauP = 0.0d0
54:       else
55:          tauP = dntauP / res_N(i-1)
56:       endif
57:       if( tauP .eq. 0.0d0 )  then
58:          alamda = 0.d0
59:       else
60:          alamda = (1.0d0 + res_N(0)*sgmT*tauP) / tauP
61:       endif
62:       omgk = S1/res_N(i-1) - alamda
63:
64: 110    ee = exp(omgk*delta_t)
65:       em = exp(-omgk*delta_t)
66:       dn11 = ee * res_N(i-1)
67:       if( omgk .eq. 0.d0 )  then
68:         dn12 = ee * ( res_N(i-1) - S1 * delta_t )
69:       else
70:         dn12 = ee * ( res_N(i-1)
71:    &          + S1 * ((1.0d0 - em)/omgk - delta_t))
72:       end if
73:
74:       Ret = abs(( dn11 - dn12 ) / dn11)
75:       if( Ret .gt. EPS ) then
76:          delta_t = delta_t * 0.5d0
77:          goto 110
78:       else
79:          res_N(i) = dn12
80:       end if
81:
82:       times_r(i) = times_r(i-1)+delta_t
83:       atimes = times_r(i) - times_r(1) + 1.0d-10
84:      write(6,'(I5,3(A,1PE12.5))') i,'  N=',res_N(i),
85:    &   '  S=',S1,'  times=',times_r(i)
86:       if( atimes .ge. drespe )  goto 120
87:     end do
88:c
89:  120 continue
90:     ntimes_r = i
91:     do i = 0, ntimes_r
92:        times_r(i) = times_r(i) / 3600d0
93:     end do
94:
95:     dnsgmT = Burn_t(1) / res_N(1)
96:     write(6,'(A,I5,2(A,1PE12.5))') 'ntimes_r=',ntimes_r,
97:    &      '  last times_r=',times_r(ntimes_r),'  dnsgmT=',dnsgmT
98:
99:
100:     return
101:
102:     end
103:
104:     subroutine respdown
105:C=================================================================
106:C      Step response -- S0 --  data prepare
107:C                        made     1998.02.09
108:C-----------------------------------------------------------------
109:C  calculate S0, N()  for  NRESPTIME(2)
110:C
111:C=================================================================
112:     implicit real*8 (A-H,O-Z)
113:C
114:     include '_sizes.h'
115:C
116:     include '_commons1.h'
117:C
118:     include 'varconst.h'
119:C
120:c...... calculate delat_t
121:
122:     sgmT = 2*4.d-16
123:     dntauP = 1.0d14
124:
125:     DNLast = res_N(1)
126:     res_N(0) = res_N(ntimes_r)
127:     res_N(1) = res_N(0)
128:c
129:     drespe = NRESPTIME(2)
130:c
131:     times_r(0) =  0.d0
132:     times_r(1) =  nstartsec + NRESPTIME(1)
133:
134:     write(6,'(4(A,1PE12.5))') '   1  N=',res_N(1),
135:    &    '  S=',S1,'  times=',times_r(1),'  Burn_t=',Burn_t(1)
136:
137:     do i = 2, MAXTIMES
138:       delta_t = drespe / 50d0
```

```
139:        if( res_N(i-1) .eq. 0.0d0 )  then
140:          tauP = 0.0d0
141:        else
142:          tauP = dntauP / res_N(i-1)
143:        endif
144:        if( tauP .eq. 0.0d0 )  then
145:          alamda = 0.d0
146:        else
147:          alamda = (1.0d0 + DNLast*sgmT*tauP) / tauP
148:        endif
149:        omgk = S0/res_N(i-1) - alamda
150: 110    ee = exp(omgk*delta_t)
151:        em = exp(-omgk*delta_t)
152:        dn11 = ee * res_N(i-1)
153:        if( omgk .eq. 0.d0 )  then
154:          dn12 = ee * ( res_N(i-1) - S0 * delta_t )
155:        else
156:          dn12 = ee * ( res_N(i-1)
157:     &            + S0 * ((1.0d0 - em)/omgk - delta_t))
158:        end if
159:        Ret = abs(( dn11 - dn12 ) / dn11)
160:        if( Ret .gt. EPS ) then
161:           delta_t = delta_t * 0.5d0
162:           goto 110
163:        else
164:           res_N(i) = dn12
165:        end if
166:
167:        times_r(i) = times_r(i-1)+delta_t
168:        atimes = times_r(i) - times_r(1) + 1.0d-10
169:        write(6,'(I5,3(A,1PE12.5))') i,'  N=',res_N(i),
170:     &     '  S=',S0,'  times=',times_r(i)
171:        if( atimes .ge. drespe )  goto 120
172:      end do
173:c
174: 120 continue
175:      ntimes_r = i
176:      do i = 0, ntimes_r
177:         times_r(i) = times_r(i) / 3600d0
178:      end do
179:
180:c     dnsgmT = Burn_t(1) / res_N(1)
181:      write(6,'(A,I5,A,1PE12.5)') 'ntimes_r=',ntimes_r,
182:     &     ' dnsgmT=',dnsgmT
183:
184:      mode = 'R--'
185:
186:      return
187:
188:      end
189:
190:      subroutine respreset ( nstartp )
191:C=====================================================================
192:C      Static data after  Step_response
193:C                      made     1998.02.09
194:C---------------------------------------------------------------------
195:C
196:C=====================================================================
197:      implicit real*8 (A-H,O-Z)
198:C
199:      include '_sizes.h'
200:C
201:      include '_commons1.h'
202:C
203:      include 'varconst.h'
204:C
205:
206:      dstartsec = ( nstartsec + NRESPTIME(1) + NRESPTIME(2)) / 3600d0
207:      do i = 1, MAXTIMES
208:        if( times(i) .gt. dstartsec ) then
209:          nstartp = i
210:          goto 100
211:        end if
212:      end do
213: 100 continue
214:      write(6,'(A,1Pe12.5,A,i5)') 'dstartsec=',dstartsec,
215:     &    ' nstartp=',nstartp
216:      do ivar = 1, NVAR
217:          XVARS(nstartp-1,ivar) = XVARS(1,ivar)
218:      end do
219:      do ip = 1, NT_PUMP
220:          Tpv_i(nstartp-1,ip)     = Tpv_i(1,ip)
221:          Rt_rem_i(nstartp-1,ip)  = Rt_rem_i(1,ip)
222:          Dpv_i(nstartp-1,ip)     = Dpv_i(1,ip)
223:          Rd_rem_i(nstartp-1,ip)  = Rd_rem_i(1,ip)
224:          Ppv_i(nstartp-1,ip)     = Ppv_i(1,ip)
225:          Rp_rem_i(nstartp-1,ip)  = Rp_rem_i(1,ip)
226:          Hepv_i(nstartp-1,ip)    = Hepv_i(1,ip)
227:          Rhe_rem_i(nstartp-1,ip) = Rhe_rem_i(1,ip)
228:      end do
229:c
230:      mode = 'RRR'
231:c
232:      return
233:c
234:      end
235:
```

● varconst.h

```
1:C ---- varconst.h
2:C
3:C === variables and constants ===
4:C
5:C    Creation date : << Tue May 6 14:21:39 1997 >>
6:C
7:C ... variables ...
8:
```

page 71

```
 9:     double precision XVARS(0:MAXTIMES,0:0)
10:C  ... XVARS can be used to refer variables with variable sequential no.
11:C  ... if necessary.
12:     common /VARIABLES/ XVARS
13:
14:C ... variable No. 1 to 20 ...
15:     double precision
16:    &      Pfusion(0:MAXTIMES), Fuelf_t(0:MAXTIMES),
17:    &      Fuel_t(0:MAXTIMES), Fuel_d(0:MAXTIMES),
18:    &      Fuel_p(0:MAXTIMES), Burn_t(0:MAXTIMES),
19:    &      Burn_d(0:MAXTIMES), Reac_p(0:MAXTIMES),
20:    &      Efusion(0:MAXTIMES), Fex_t(0:MAXTIMES),
21:    &      Fex_d(0:MAXTIMES), Fex_p(0:MAXTIMES), Fhe_ex(0:MAXTIMES),
22:    &      i_t(0:MAXTIMES), i_d(0:MAXTIMES), i_p(0:MAXTIMES),
23:    &      Adiv_t(0:MAXTIMES), Afw_t(0:MAXTIMES),
24:    &      d_Adiv_t(0:MAXTIMES), d_Afw_t(0:MAXTIMES)
25:     common /VARIABLES/
26:    &      Pfusion, Fuelf_t, Fuel_t, Fuel_d, Fuel_p, Burn_t, Burn_d,
27:    &      Reac_p, Efusion, Fex_t, Fex_d, Fex_p, Fhe_ex, i_t, i_d,
28:    &      i_p, Adiv_t, Afw_t, d_Adiv_t, d_Afw_t
29:
30:C ... variable No. 21 to 40 ...
31:     double precision
32:    &      C_t(0:MAXTIMES), d_C_t(0:MAXTIMES), D_t(0:MAXTIMES),
33:    &      d_D_t(0:MAXTIMES), Adiv_d(0:MAXTIMES), Afw_d(0:MAXTIMES),
34:    &      d_Adiv_d(0:MAXTIMES), d_Afw_d(0:MAXTIMES),
35:    &      C_d(0:MAXTIMES), d_C_d(0:MAXTIMES), D_d(0:MAXTIMES),
36:    &      d_D_d(0:MAXTIMES), Adiv_p(0:MAXTIMES), Afw_p(0:MAXTIMES),
37:    &      d_Adiv_p(0:MAXTIMES), d_Afw_p(0:MAXTIMES),
38:    &      C_p(0:MAXTIMES), d_C_p(0:MAXTIMES), D_p(0:MAXTIMES),
39:    &      d_D_p(0:MAXTIMES)
40:     common /VARIABLES/
41:    &      C_t, d_C_t, D_t, d_D_t, Adiv_d, Afw_d, d_Adiv_d, d_Afw_d,
42:    &      C_d, d_C_d, D_d, d_D_d, Adiv_p, Afw_p, d_Adiv_p, d_Afw_p,
43:    &      C_p, d_C_p, D_p, d_D_p
44:
45:C ... variable No. 41 to 60 ...
46:     double precision
47:    &      Tpfc_(0:MAXTIMES), Tpfc_m(0:MAXTIMES), Dpfc_(0:MAXTIMES),
48:    &      Ppfc_(0:MAXTIMES), Tpv(0:MAXTIMES), Tp_reg(0:MAXTIMES),
49:    &      Tp_tot(0:MAXTIMES), Tp_tot_m(0:MAXTIMES),
50:    &      Rt_rem(0:MAXTIMES), Dpv(0:MAXTIMES), Rd_rem(0:MAXTIMES),
51:    &      Dp_reg(0:MAXTIMES), Dp_tot(0:MAXTIMES),
52:    &      Rp_rem(0:MAXTIMES), Pp_reg(0:MAXTIMES),
53:    &      Pp_tot(0:MAXTIMES), Ppv(0:MAXTIMES), Rhe_rem(0:MAXTIMES),
54:    &      Hepv(0:MAXTIMES), Hep_reg(0:MAXTIMES)
55:     common /VARIABLES/
56:    &      Tpfc_, Tpfc_m, Dpfc_, Ppfc_, Tpv, Tp_reg, Tp_tot,
57:    &      Tp_tot_m, Rt_rem, Dpv, Rd_rem, Dp_reg, Dp_tot, Rp_rem,
58:    &      Pp_reg, Pp_tot, Ppv, Rhe_rem, Hepv, Hep_reg
59:
60:C ... variable No. 61 to 80 ...
61:     double precision
62:    &      Hep_tot(0:MAXTIMES), Fv_out(0:MAXTIMES),
63:    &      Fpuri_in(0:MAXTIMES), it_(0:MAXTIMES), id_(0:MAXTIMES),
64:    &      ip_(0:MAXTIMES), xnhydro_(0:MAXTIMES),
65:    &      xh_imp(0:MAXTIMES), Fadj_puri(0:MAXTIMES),
66:    &      Fhydro_pure(0:MAXTIMES), Fhydro_imp(0:MAXTIMES),
67:    &      Fnhydro_imp(0:MAXTIMES), Ft_pure(0:MAXTIMES),
68:    &      Fd_pure(0:MAXTIMES), Fp_pure(0:MAXTIMES),
69:    &      Fcq4_imp(0:MAXTIMES), Fnq3_imp(0:MAXTIMES),
70:    &      Fq2o_imp(0:MAXTIMES), Fhydro_imp_w(0:MAXTIMES),
71:    &      f_hydro_imp_w(0:MAXTIMES)
72:     common /VARIABLES/
73:    &      Hep_tot, Fv_out, Fpuri_in, it_, id_, ip_, xnhydro_,
74:    &      xh_imp, Fadj_puri, Fhydro_pure, Fhydro_imp, Fnhydro_imp,
75:    &      Ft_pure, Fd_pure, Fp_pure, Fcq4_imp, Fnq3_imp, Fq2o_imp,
76:    &      Fhydro_imp_w, f_hydro_imp_w
77:
78:C ... variable No. 81 to 100 ...
79:     double precision
80:    &      Nimpt(0:MAXTIMES), Nhydro_imp(0:MAXTIMES),
81:    &      Nnhydro_imp(0:MAXTIMES), Ncq4_imp(0:MAXTIMES),
82:    &      Nnq3_imp(0:MAXTIMES), Nq2o_imp(0:MAXTIMES),
83:    &      Timp_(0:MAXTIMES), Dimp_(0:MAXTIMES), Pimp_(0:MAXTIMES),
84:    &      Fh2_puri_in(0:MAXTIMES), Ft2_sub_in(0:MAXTIMES),
85:    &      Fd2_sub_in(0:MAXTIMES), Fp2_sub_in(0:MAXTIMES),
86:    &      Fh2_sub_in(0:MAXTIMES), Ns(0:MAXTIMES),
87:    &      xt_sub(0:MAXTIMES), xd_sub(0:MAXTIMES),
88:    &      xp_sub(0:MAXTIMES), T2_sub(0:MAXTIMES),
89:    &      D2_sub(0:MAXTIMES)
90:     common /VARIABLES/
91:    &      Nimpt, Nhydro_imp, Nnhydro_imp, Ncq4_imp, Nnq3_imp,
92:    &      Nq2o_imp, Timp_, Dimp_, Pimp_, Fh2_puri_in, Ft2_sub_in,
93:    &      Fd2_sub_in, Fp2_sub_in, Fh2_sub_in, Ns, xt_sub, xd_sub,
94:    &      xp_sub, T2_sub, D2_sub
95:
96:C ... variable No. 101 to 120 ...
97:     double precision
98:    &      P2_sub(0:MAXTIMES), Nhydro_sub(0:MAXTIMES),
99:    &      Ft_sub_out(0:MAXTIMES), Fd_sub_out(0:MAXTIMES),
100:   &      Fp_sub_out(0:MAXTIMES), Fh_sub_out(0:MAXTIMES),
101:   &      Treg_out(0:MAXTIMES), Ecq4_reg(0:MAXTIMES),
102:   &      Enq3_reg(0:MAXTIMES), Eq2o_reg(0:MAXTIMES),
103:   &      Ncq4_reg(0:MAXTIMES), Nnq3_reg(0:MAXTIMES),
104:   &      Nq2o_reg(0:MAXTIMES), Fhydro_pmem(0:MAXTIMES),
105:   &      Tpmem(0:MAXTIMES), Dpmem(0:MAXTIMES), Ppmem(0:MAXTIMES),
106:   &      it_endj(0:MAXTIMES), id_endj(0:MAXTIMES),
107:   &      ip_endj(0:MAXTIMES)
108:    common /VARIABLES/
109:   &      P2_sub, Nhydro_sub, Ft_sub_out, Fd_sub_out, Fp_sub_out,
110:   &      Fh_sub_out, Treg_out, Ecq4_reg, Enq3_reg, Eq2o_reg,
111:   &      Ncq4_reg, Nnq3_reg, Nq2o_reg, Fhydro_pmem, Tpmem, Dpmem,
112:   &      Ppmem, it_endj, id_endj, ip_endj
113:
114:C ... variable No. 121 to 140 ...
115:    double precision
116:   &      Ft_puri_out(0:MAXTIMES), Fd_puri_out(0:MAXTIMES),
117:   &      Fp_puri_out(0:MAXTIMES), Fpuri_out(0:MAXTIMES),
118:   &      Xiss_in_t(0:MAXTIMES), Xiss_in_d(0:MAXTIMES),
119:   &      Xiss_in_p(0:MAXTIMES), Fpure_(0:MAXTIMES),
120:   &      Fimp_(0:MAXTIMES), Timp_m(0:MAXTIMES),
121:   &      Treg_out_m(0:MAXTIMES), Fiss_in(0:MAXTIMES),
122:   &      Fiss_nb(0:MAXTIMES), Fiss_in_t(0:MAXTIMES),
```

```
     &      Xiss_t(0:MAXTIMES), Xnb_t(0:MAXTIMES),
     &      Xinj_t(0:MAXTIMES), Fiss_in_d(0:MAXTIMES),
     &      Xiss_d(0:MAXTIMES), Xnb_d(0:MAXTIMES)
      common /VARIABLES/
     &      Ft_puri_out, Fd_puri_out, Fp_puri_out, Fpuri_out,
     &      Xiss_in_t, Xiss_in_d, Xiss_in_p, Fpure_, Fimp_, Timp_m,
     &      Treg_out_m, Fiss_in, Fiss_nb, Fiss_in_t, Xiss_t, Xnb_t,
     &      Xinj_t, Fiss_in_d, Xiss_d, Xnb_d

C ... variable No. 141 to 160 ...
      double precision
     &      Xinj_d(0:MAXTIMES), Fiss_in_p(0:MAXTIMES),
     &      Niss_(0:MAXTIMES), Tiss_(0:MAXTIMES), Tiss_m(0:MAXTIMES),
     &      Diss_(0:MAXTIMES), Piss_(0:MAXTIMES), Xiss_p(0:MAXTIMES),
     &      Fiss_top_(0:MAXTIMES), Fiss_mid_(0:MAXTIMES),
     &      Fiss_bot_(0:MAXTIMES), Fiss_inj(0:MAXTIMES),
     &      Fplas_nb_d(0:MAXTIMES), Fin_st_d(0:MAXTIMES),
     &      Fin_rec_(0:MAXTIMES), Fin_iss_mid(0:MAXTIMES),
     &      Xcryo_d(0:MAXTIMES), W_nb(0:MAXTIMES),
     &      Ncryo_(0:MAXTIMES), Fbk_nb_(0:MAXTIMES)
      common /VARIABLES/
     &      Xinj_d, Fiss_in_p, Niss_, Tiss_, Tiss_m, Diss_, Piss_,
     &      Xiss_p, Fiss_top_, Fiss_mid_, Fiss_bot_, Fiss_inj,
     &      Fplas_nb_d, Fin_st_d, Fin_rec_, Fin_iss_mid, Xcryo_d,
     &      W_nb, Ncryo_, Fbk_nb_

C ... variable No. 161 to 180 ...
      double precision
     &      Dcryo_(0:MAXTIMES), Dnb_reg_(0:MAXTIMES),
     &      Tnb_cryo(0:MAXTIMES), Fbk_nb_t(0:MAXTIMES),
     &      Rnb_reg_t(0:MAXTIMES), tnb_(0:MAXTIMES),
     &      Tnb_reg_(0:MAXTIMES), Tnb_cryo_tot(0:MAXTIMES),
     &      Tnb_cryo_mob(0:MAXTIMES), Dnb_cryo(0:MAXTIMES),
     &      Fbk_nb_d(0:MAXTIMES), Rnb_reg_d(0:MAXTIMES),
     &      Dnb_cryo_tot(0:MAXTIMES), Xnb_cryo_t(0:MAXTIMES),
     &      Xnb_cryo_d(0:MAXTIMES), Xnb_cryo_p(0:MAXTIMES),
     &      Pnb_cryo(0:MAXTIMES), Tnb_wall(0:MAXTIMES),
     &      Tnb_wall_mob(0:MAXTIMES), Tnb_tot(0:MAXTIMES)
      common /VARIABLES/
     &      Dcryo_, Dnb_reg_, Tnb_cryo, Fbk_nb_t, Rnb_reg_t, tnb_,
     &      Tnb_reg_, Tnb_cryo_tot, Tnb_cryo_mob, Dnb_cryo, Fbk_nb_d,
     &      Rnb_reg_d, Dnb_cryo_tot, Xnb_cryo_t, Xnb_cryo_d,
     &      Xnb_cryo_p, Pnb_cryo, Tnb_wall, Tnb_wall_mob, Tnb_tot

C ... variable No. 181 to 200 ...
      double precision
     &      Tnb_tot_mob(0:MAXTIMES), Fnb_cryo(0:MAXTIMES),
     &      Crec(0:MAXTIMES), Fplas_nb_(0:MAXTIMES),
     &      Fplas_nb_t(0:MAXTIMES), Fplas_nb_p(0:MAXTIMES),
     &      ncycle_p(0:MAXTIMES), Nnb_cryo(0:MAXTIMES),
     &      Rnb_reg_(0:MAXTIMES), Rnb_reg_p(0:MAXTIMES),
     &      Fgas_t(0:MAXTIMES), Fgas_in_tri(0:MAXTIMES),
     &      Fgas_in_deu(0:MAXTIMES), Fgas_d(0:MAXTIMES),
     &      Ngas_tot(0:MAXTIMES), Fgas_tot(0:MAXTIMES),
     &      Tgas_(0:MAXTIMES), Dgas_(0:MAXTIMES), Xgas_t(0:MAXTIMES),
     &      Xgas_d(0:MAXTIMES)
      common /VARIABLES/
     &      Tnb_tot_mob, Fnb_cryo, Crec, Fplas_nb_, Fplas_nb_t,
     &      Fplas_nb_p, ncycle_p, Nnb_cryo, Rnb_reg_, Rnb_reg_p,
     &      Fgas_t, Fgas_in_tri, Fgas_in_deu, Fgas_d, Ngas_tot,
     &      Fgas_tot, Tgas_, Dgas_, Xgas_t, Xgas_d

C ... variable No. 201 to 220 ...
      double precision
     &      Xgas_p(0:MAXTIMES), Pgas_(0:MAXTIMES),
     &      Fpuff_plas(0:MAXTIMES), Tgas_mob(0:MAXTIMES),
     &      Fplas_in_t(0:MAXTIMES), Fplas_in_d(0:MAXTIMES),
     &      Fplas_in_p(0:MAXTIMES), Fsto_out_t(0:MAXTIMES),
     &      Fsto_out_d(0:MAXTIMES), Fsto_out_p(0:MAXTIMES),
     &      Next_12(0:MAXTIMES), Text_12(0:MAXTIMES),
     &      Dext_12(0:MAXTIMES), Pext_12(0:MAXTIMES),
     &      Xext_12_t(0:MAXTIMES), Xext_12_d(0:MAXTIMES),
     &      Xext_12_p(0:MAXTIMES), Next_tot(0:MAXTIMES),
     &      Text_tot(0:MAXTIMES), Dext_tot(0:MAXTIMES)
      common /VARIABLES/
     &      Xgas_p, Pgas_, Fpuff_plas, Tgas_mob, Fplas_in_t,
     &      Fplas_in_d, Fplas_in_p, Fsto_out_t, Fsto_out_d,
     &      Fsto_out_p, Next_12, Text_12, Dext_12, Pext_12,
     &      Xext_12_t, Xext_12_d, Xext_12_p, Next_tot, Text_tot,
     &      Dext_tot

C ... variable No. 221 to 240 ...
      double precision
     &      Pext_tot(0:MAXTIMES), Text_m(0:MAXTIMES),
     &      Vinj_t(0:MAXTIMES), Vinj_d(0:MAXTIMES),
     &      Vinj_p(0:MAXTIMES), Xextru_t(0:MAXTIMES),
     &      Xextru_d(0:MAXTIMES), Xextru_p(0:MAXTIMES),
     &      Nextru_(0:MAXTIMES), Vinj_tot(0:MAXTIMES),
     &      Dpellet(0:MAXTIMES), Hpellet(0:MAXTIMES),
     &      npellet_t(0:MAXTIMES), npellet_d(0:MAXTIMES),
     &      npellet_p(0:MAXTIMES), Rinj_(0:MAXTIMES),
     &      Finj_in_tot(0:MAXTIMES), Fbk_plas_(0:MAXTIMES),
     &      Fbk_plas_t(0:MAXTIMES), Fbk_plas_d(0:MAXTIMES)
      common /VARIABLES/
     &      Pext_tot, Text_m, Vinj_t, Vinj_d, Vinj_p, Xextru_t,
     &      Xextru_d, Xextru_p, Nextru_, Vinj_tot, Dpellet, Hpellet,
     &      npellet_t, npellet_d, npellet_p, Rinj_, Finj_in_tot,
     &      Fbk_plas_, Fbk_plas_t, Fbk_plas_d

C ... variable No. 241 to 260 ...
      double precision
     &      Fbk_plas_p(0:MAXTIMES), Fplas_prop(0:MAXTIMES),
     &      Fplas_prop_t(0:MAXTIMES), Fplas_prop_d(0:MAXTIMES),
     &      Fplas_prop_p(0:MAXTIMES), Finj_exh(0:MAXTIMES),
     &      Finj_exh_t(0:MAXTIMES), Finj_exh_d(0:MAXTIMES),
     &      Xinj_p(0:MAXTIMES), Finj_prop(0:MAXTIMES),
     &      Xprop_t(0:MAXTIMES), Xprop_d(0:MAXTIMES),
     &      Xprop_p(0:MAXTIMES), Finj_rec(0:MAXTIMES),
     &      Fsto_tri_in(0:MAXTIMES), Fsto_tri_out(0:MAXTIMES),
     &      Nsto_tri_(0:MAXTIMES), Tsto_tri_(0:MAXTIMES),
     &      Tsto_tri_m(0:MAXTIMES), Dsto_tri_(0:MAXTIMES)
      common /VARIABLES/
     &      Fbk_plas_p, Fplas_prop, Fplas_prop_t, Fplas_prop_d,
     &      Fplas_prop_p, Finj_exh, Finj_exh_t, Finj_exh_d, Xinj_p,
```

page 73

```
237:    &       Finj_prop, Xprop_t, Xprop_d, Xprop_p, Finj_rec,
238:    &       Fsto_tri_in, Fsto_tri_out, Nsto_tri_, Tsto_tri_,
239:    &       Tsto_tri_m, Dsto_tri_
240:
241:C ... variable No. 261 to 280 ...
242:     double precision
243:    &       Psto_tri_(0:MAXTIMES), Xsto_tri_t(0:MAXTIMES),
244:    &       Xsto_tri_d(0:MAXTIMES), Xsto_tri_p(0:MAXTIMES),
245:    &       Fsto_deu_in(0:MAXTIMES), Fsto_deu_out(0:MAXTIMES),
246:    &       Nsto_deu_(0:MAXTIMES), Tsto_deu_(0:MAXTIMES),
247:    &       Tsto_deu_m(0:MAXTIMES), Dsto_deu_(0:MAXTIMES),
248:    &       Psto_deu_(0:MAXTIMES), Ksto_tri_(0:MAXTIMES),
249:    &       Ksto_deu_(0:MAXTIMES), Ksto_pro_(0:MAXTIMES),
250:    &       Xsto_deu_t(0:MAXTIMES), Xsto_deu_d(0:MAXTIMES),
251:    &       Xsto_deu_p(0:MAXTIMES), Fsto_pro_in(0:MAXTIMES),
252:    &       Fsto_pro_out(0:MAXTIMES), Finj_sto_pro(0:MAXTIMES)
253:     common /VARIABLES/
254:    &       Psto_tri_, Xsto_tri_t, Xsto_tri_d, Xsto_tri_p,
255:    &       Fsto_deu_in, Fsto_deu_out, Nsto_deu_, Tsto_deu_,
256:    &       Tsto_deu_m, Dsto_deu_, Psto_deu_, Ksto_tri_, Ksto_deu_,
257:    &       Ksto_pro_, Xsto_deu_t, Xsto_deu_d, Xsto_deu_p,
258:    &       Fsto_pro_in, Fsto_pro_out, Finj_sto_pro
259:
260:C ... variable No. 281 to 291 ...
261:     double precision
262:    &       Nsto_pro_(0:MAXTIMES), Tsto_pro_(0:MAXTIMES),
263:    &       Tsto_pro_m(0:MAXTIMES), Dsto_pro_(0:MAXTIMES),
264:    &       Psto_pro_(0:MAXTIMES), Xsto_pro_t(0:MAXTIMES),
265:    &       Xsto_pro_d(0:MAXTIMES), Xsto_pro_p(0:MAXTIMES),
266:    &       Fsto_tri_out_t(0:MAXTIMES), Fsto_deu_out_d(0:MAXTIMES),
267:    &       Fsto_pro_out_p(0:MAXTIMES)
268:     common /VARIABLES/
269:    &       Nsto_pro_, Tsto_pro_, Tsto_pro_m, Dsto_pro_, Psto_pro_,
270:    &       Xsto_pro_t, Xsto_pro_d, Xsto_pro_p, Fsto_tri_out_t,
271:    &       Fsto_deu_out_d, Fsto_pro_out_p
272:
273:     common /VARIABLES/ VARNAME(MAXVAR),VARDESC(MAXVAR),
274:    &        IVARFLAG(MAXVAR)
275:
276:     integer NVAR
277:     common /NVARIABLS/ NVAR
278:
279:     character*32 VARNAME
280:     character*64 VARDESC
281:     integer IVARFLAG
282:
283:C
284:     double precision C(0:0)
285:C  ... C can be used to refer constants with sequential no.
286:C  ... if necessary.
287:     common /CONSTANTS/ C
288:
289:C ... constants ...
290:
291:     double precision ap
292:     common /CONSTANTS/ ap
293:     double precision fbu
294:     common /CONSTANTS/ fbu
295:     double precision fin_pellet
296:     common /CONSTANTS/ fin_pellet
297:     double precision fin_gas
298:     common /CONSTANTS/ fin_gas
299:     double precision Temperature
300:     common /CONSTANTS/ Temperature
301:     double precision Tedge
302:     common /CONSTANTS/ Tedge
303:     double precision Medge
304:     common /CONSTANTS/ Medge
305:     double precision mu_A
306:     common /CONSTANTS/ mu_A
307:     double precision mu_C
308:     common /CONSTANTS/ mu_C
309:     double precision mu_D
310:     common /CONSTANTS/ mu_D
311:     double precision dc
312:     common /CONSTANTS/ dc
313:     double precision Vdiv
314:     common /CONSTANTS/ Vdiv
315:     double precision Vfw
316:     common /CONSTANTS/ Vfw
317:     double precision Ad
318:     common /CONSTANTS/ Ad
319:     double precision lambda_T
320:     common /CONSTANTS/ lambda_T
321:     double precision npump
322:     common /CONSTANTS/ npump
323:     double precision treg
324:     common /CONSTANTS/ treg
325:     double precision tpump
326:     common /CONSTANTS/ tpump
327:     double precision emr
328:     common /CONSTANTS/ emr
329:     double precision Rs
330:     common /CONSTANTS/ Rs
331:     double precision Area_p
332:     common /CONSTANTS/ Area_p
333:     double precision mu_v
334:     common /CONSTANTS/ mu_v
335:     double precision mu_mob
336:     common /CONSTANTS/ mu_mob
337:     double precision Lhydro_
338:     common /CONSTANTS/ Lhydro_
339:     double precision Limp_
340:     common /CONSTANTS/ Limp_
341:     double precision Ns_max
342:     common /CONSTANTS/ Ns_max
343:     double precision treg_msieve
344:     common /CONSTANTS/ treg_msieve
345:     double precision mu_reg_m
346:     common /CONSTANTS/ mu_reg_m
347:     double precision tau_pmem
348:     common /CONSTANTS/ tau_pmem
349:     double precision wl
350:     common /CONSTANTS/ wl
```

```
351:    double precision w2
352:    common /CONSTANTS/ w2
353:    double precision w3
354:    common /CONSTANTS/ w3
355:    double precision xcq4_
356:    common /CONSTANTS/ xcq4_
357:    double precision xnq3_
358:    common /CONSTANTS/ xnq3_
359:    double precision xq2o_
360:    common /CONSTANTS/ xq2o_
361:    double precision Fbl_cool
362:    common /CONSTANTS/ Fbl_cool
363:    double precision Fbl_
364:    common /CONSTANTS/ Fbl_
365:    double precision h1
366:    common /CONSTANTS/ h1
367:    double precision h3
368:    common /CONSTANTS/ h3
369:    double precision Fragnb
370:    common /CONSTANTS/ Fragnb
371:    double precision Fragiss_inj
372:    common /CONSTANTS/ Fragiss_inj
373:    double precision Xc_t
374:    common /CONSTANTS/ Xc_t
375:    double precision Xbl_t
376:    common /CONSTANTS/ Xbl_t
377:    double precision Xc_d
378:    common /CONSTANTS/ Xc_d
379:    double precision Xbl_d
380:    common /CONSTANTS/ Xbl_d
381:    double precision tau_iss_
382:    common /CONSTANTS/ tau_iss_
383:    double precision mu_iss_
384:    common /CONSTANTS/ mu_iss_
385:    double precision Xiss_top_t
386:    common /CONSTANTS/ Xiss_top_t
387:    double precision Xiss_top_d
388:    common /CONSTANTS/ Xiss_top_d
389:    double precision Xiss_top_p
390:    common /CONSTANTS/ Xiss_top_p
391:    double precision Xiss_mid_t
392:    common /CONSTANTS/ Xiss_mid_t
393:    double precision Xiss_mid_d
394:    common /CONSTANTS/ Xiss_mid_d
395:    double precision Xiss_mid_p
396:    common /CONSTANTS/ Xiss_mid_p
397:    double precision Xiss_bot_t
398:    common /CONSTANTS/ Xiss_bot_t
399:    double precision Xiss_bot_d
400:    common /CONSTANTS/ Xiss_bot_d
401:    double precision Xiss_bot_p
402:    common /CONSTANTS/ Xiss_bot_p
403:    double precision eta_nb_tot
404:    common /CONSTANTS/ eta_nb_tot
405:    double precision kiss_mid_
406:    common /CONSTANTS/ kiss_mid_
407:    double precision fcryo_
408:    common /CONSTANTS/ fcryo_
409:    double precision tnb_reg
410:    common /CONSTANTS/ tnb_reg
411:    double precision tnb_pump
412:    common /CONSTANTS/ tnb_pump
413:    double precision nnb_
414:    common /CONSTANTS/ nnb_
415:    double precision mu_nb_mob
416:    common /CONSTANTS/ mu_nb_mob
417:    double precision fnb_wall
418:    common /CONSTANTS/ fnb_wall
419:    double precision mu_wall_
420:    common /CONSTANTS/ mu_wall_
421:    double precision tcycle_p
422:    common /CONSTANTS/ tcycle_p
423:    double precision t_p_ru
424:    common /CONSTANTS/ t_p_ru
425:    double precision t_p_rd
426:    common /CONSTANTS/ t_p_rd
427:    double precision t_nb_ru
428:    common /CONSTANTS/ t_nb_ru
429:    double precision t_nb_rd
430:    common /CONSTANTS/ t_nb_rd
431:    double precision t_burn
432:    common /CONSTANTS/ t_burn
433:    double precision tcycle_nb
434:    common /CONSTANTS/ tcycle_nb
435:    double precision E_nb
436:    common /CONSTANTS/ E_nb
437:    double precision No
438:    common /CONSTANTS/ No
439:    double precision nedge_tot
440:    common /CONSTANTS/ nedge_tot
441:    double precision nedge_t
442:    common /CONSTANTS/ nedge_t
443:    double precision nedge_d
444:    common /CONSTANTS/ nedge_d
445:    double precision nedge_p
446:    common /CONSTANTS/ nedge_p
447:    double precision mgas_mid_iss
448:    common /CONSTANTS/ mgas_mid_iss
449:    double precision mgas_bot_iss
450:    common /CONSTANTS/ mgas_bot_iss
451:    double precision mu_gas
452:    common /CONSTANTS/ mu_gas
453:    double precision Vpellet
454:    common /CONSTANTS/ Vpellet
455:    double precision finj_err
456:    common /CONSTANTS/ finj_err
457:    double precision linj_mid_iss
458:    common /CONSTANTS/ linj_mid_iss
459:    double precision linj_bot_iss
460:    common /CONSTANTS/ linj_bot_iss
461:    double precision mu_ext
462:    common /CONSTANTS/ mu_ext
463:    double precision kpellet_t
464:    common /CONSTANTS/ kpellet_t
```

```
465:     double precision kpellet_d
466:     common /CONSTANTS/ kpellet_d
467:     double precision kpellet_p
468:     common /CONSTANTS/ kpellet_p
469:     double precision fcap_prop
470:     common /CONSTANTS/ fcap_prop
471:     double precision fnon_iss
472:     common /CONSTANTS/ fnon_iss
473:     double precision liss_p
474:     common /CONSTANTS/ liss_p
475:     double precision nprop
476:     common /CONSTANTS/ nprop
477:     double precision mu_sto
478:     common /CONSTANTS/ mu_sto
479:     double precision linj_p
480:     common /CONSTANTS/ linj_p
481:     double precision t_dwell
482:     common /CONSTANTS/ t_dwell
483:     double precision volume
484:     common /CONSTANTS/ volume
485:     double precision Namsb_max
486:     common /CONSTANTS/ Namsb_max
487:     double precision Ncmsb_max
488:     common /CONSTANTS/ Ncmsb_max
489:
490:     integer  NCONST
491:
492:     common /NCONSTANT/ NCONST
493:     common /CONSTANTS/ CONSTNAME(MAXCONST),CONSTDESC(MAXCONST)
494:
495:     character*32 CONSTNAME
496:     character*64 CONSTDESC
497:
498:C ... dX/dt
499:
500:     parameter ( NDFVAR = 47)
501:
502:     common /BLOCK3/ IDFVAR(NDFVAR)
503:
504:     integer IDFVAR, iabxctl
505:
506:C ... pump
507:
508:     parameter ( NT_PUMP = 24)
509:
510:     double precision
511:    & Tv(NT_PUMP),
512:    & Tpv_i(0:MAXTIMES,NT_PUMP), Rt_rem_i(0:MAXTIMES,NT_PUMP),
513:    & Dpv_i(0:MAXTIMES,NT_PUMP), Rd_rem_i(0:MAXTIMES,NT_PUMP),
514:    & Ppv_i(0:MAXTIMES,NT_PUMP), Rp_rem_i(0:MAXTIMES,NT_PUMP),
515:    & Hepv_i(0:MAXTIMES,NT_PUMP), Rhe_rem_i(0:MAXTIMES,NT_PUMP)
516:C   & , Tv_s(NT_PUMP), Tp_reg_i(NT_PUMP), Dp_reg_i(NT_PUMP),
517:C   & Pp_reg_i(NT_PUMP), Hep_reg_i(NT_PUMP)
518:     common / PUMPVAR / Tv,
519:    & Tpv_i, Rt_rem_i, Dpv_i, Rd_rem_i,
520:    & Ppv_i, Rp_rem_i, Hepv_i, Rhe_rem_i
521:C   & , Tv_s, Tp_reg_i, Dp_reg_i, Pp_reg_i, Hep_reg_i
522:     common / IPUMPVAR / iabxctl
523:
524:C ...  condition check
525:     integer mcycle(3)
526:     double precision stime(3), scondvar(3)
527:     common / NORM / stime, scondvar
528:     common / MNORM / mcycle
529:
530:c .... step response
531:     double precision res_N(0:MAXTIMES), S0, S1
532:     common / STEPR / res_N, S0, S1
533:
534:c .... tend
535:     double precision tend, Timp_tend
536:     common / FCLEAN / tend, Timp_tend
537:
538:C ---- end of varconst.h
```

● _commons1.h

```
 1:C
 2:C ------------------------------------------------------------------
 3:C  Problem control variables
 4:C ------------------------------------------------------------------
 5:C------------------------------------------------------------------
 6:C $Id: _commons1.h,v 1.6 1997/04/21 08:53:12 sasaki Exp sasaki $
 7:C------------------------------------------------------------------
 8:C
 9:     character*128 print
10:     character*128 result
11:     character*128 result_o
12:     character*32 sysname
13:     character*128 response
14:     common /FILES/ print, result, sysname, response
15:     common /FILESI/ llp, llr
16:C*****************************************************************
17:C*   ... file names
18:C*
19:C*    PRINT : standard output ( I/O unit 6 )
20:C*    RESULT : directory for result output  (I/O unit 10)
21:C*    SYSNAME : system type on which this program is running
22:C*            (Machintosh/UNIX/Windows/....)
23:C*    RESPONSE : response file name
24:C*
25:C*    LLP: length of effective part as name in PRINT
26:C*    LLR: length of effective part as name in RESULT
27:C*
28:C*****************************************************************
29:     character*32 METHOD
30:     character*8  MODE
31:     integer JDEBUG, OUTPUTALL, JDISP
```

```
32:     common /OPTION/ METHOD, JDEBUG, OUTPUTALL, MODE, JDISP
33:C*****************************************************************
34:C*   ... options
35:C*
36:C*    METHOD : method of solution
37:C*        'RK' or ' ' : 4th order Runge-Kutta method.
38:C*        'EULER' : Explicit euler method.
39:C*    JDEBUG : debug output if nonzero.
40:C*    JDISP  : display to screen  cycle,step
41:C*    MODE   : 'RUN'  or 'RES'
42:C*
43:C*****************************************************************
44:CCC   parameter ( MAXTIMES = 1500, MAXTZONE = 32 )
45:C
46:     integer ntimes, ntimezone, nsteps,nsubsteps, ncycle,
47:    &      nstartc, nstartsec, nbreakc, nbreakp, ntimes_r
48:     common /CONTROL/ ntimes, ntimezone, nsteps(MAXTZONE),
49:    &             nsubsteps(MAXTZONE), ncycle,
50:    &             nstartc, nstartsec,
51:    &             nbreakc(MAXRES), nbreakp(MAXRES),
52:    &             ntimes_r
53:C
54:C*****************************************************************
55:C*   ... number of time steps etc.
56:C*
57:C*   NTIMES  : total number of calculation time steps.
58:C*
59:C*   NTIMEZONE  : total number of time zones.
60:C*          "Time zone" is a time range during which time steps
61:C*          have the same witdh.
62:C*          When ntimezone=0, all time meshs have the same width
63:C*          from t=0 to TMAX,
64:C*
65:C*   NSTEPS(NTIMEZONE)  : number of time steps of each time zone.
66:C*   NSUBSTEPS(NTIMEZONE) : number of sub-time steps of each time zone.
67:C*   NSTARTC : start point cycle No.
68:C*   NSTARTP :          step No.
69:C*   NBREAKC : break point cycle No.
70:C*   NBREAKP :          sec     ---> step No.
71:C*
72:C*****************************************************************
73:C
74:     double precision t, tmax,delta_t,times,tzbound,
75:    &             times_r, dbreakp, dnsgmT
76:     common /DCONTROL/ t, tmax,delta_t,times(0:MAXTIMES),
77:    &             tzbound(0:MAXTZONE),times_r(0:MAXTIMES),
78:    &             dbreakp(MAXRES), dnsgmT
79:C
80:C*****************************************************************
81:C*   ... floating point data
82:C*
83:C*   T     : current time
84:C*   TMAX  : max simulation time
85:C*   DELTA_T  : current time step width
86:C*   TIMES(0:NTIMES)  : time step boundaries
87:C*   TZNOUND(0:NTIMES)  : time zone boundaries
88:C*
89:C*****************************************************************
90:C
91:     parameter (maxvarout = 300)
92:     character*32 varout
93:     common /NMONITOR/ varout(maxvarout)
94:     integer nvarout, ivarout
95:     common /VMONITOR/ nvarout, ivarout(maxvarout)
96:C*****************************************************************
97:C*   ... Variables name of monitored variables etc.
98:C*
99:C*   NVAROUT  : number of variables whose value on each time step
100:C*            are output as results.
101:C*   VAROUT(NVAROUT) : name of variable whose value is output.
102:C*   IVAROUT(NVAROUT) : variable position no. in varname(nvar)
103:C*****************************************************************
104:C
105:     integer nrespst, nresptime
106:     double precision deltas, eps
107:     common /DRESP/ deltas, eps
108:     common /NRESP/ nrespst(2), nresptime(2)
109:C*****************************************************************
110:C*   NRESPST   (1):step response start cycle
111:C*            (2):                sec
112:C*   NRESPTIME (1):continuous time power S1 (sec)
113:C*            (2):                  S0     after (1)
114:C*   DELTAS
115:C*   EPS
116:C*****************************************************************
```

● _storage.h

```
1:      parameter (MAXSTORAGE= 50000)
2:      double precision D
3:      common /STORAGE/ D(MAXSTORAGE), LIMIT
4:C-----------------------------------------------------------------
5:C $Id: _storage.h,v 1.3 1997/04/06 05:51:58 sasaki Exp sasaki $
6:C*****************************************************************
7:C*   ...  data area to store calculated variable values etc.
8:C*
9:C*   D(*) : double precision memory area
10:C*   LIMIT : size of arry D(*)
11:C*
12:C*****************************************************************
13:C
```

page 77

● _sizes.h

```
C
C  ... Problem size parameters
C
C $Id: _sizes.h,v 1.3 1997/04/07 04:35:29 sasaki Exp sasaki $
C
      parameter ( MAXTIMES = 2000 )
      parameter ( MAXTZONE = 64 )
      parameter ( MAXRES = 10 )
      parameter ( MAXCONST = 100 )
      parameter ( MAXVAR = 291 )
```