Chap B. Genesis : General Neural Simulation System

Plan

I. Genesis : un progiciel pour la simulation neuronale

Après avoir indiqué les bases concernant les aspects numériques des simulations, on va entreprendre l'apprentissage du progiciel et de son langage à l'aide de quelques exemples. Genesis signifie General Neural SImulation System et a été développé au Californian Institute of Technology (Caltech), USA, depuis les années 85. On va donner maintenant une introduction aux exemples (ou tutorials) ainsi qu'une démonstration de l'utilisation de l'interface graphique.

Genesis est ce que l'on appelle un simulateur généraliste qui doit être distingué des simulateurs dits dédiés. Ces derniers, écrits dans un langage de programmation tels que C par exemple, réalisent des taches particulièrement bien ciblées. Les codes numériques obtenus peuvent donner lieu à des éxécutions plus rapides que celles obtenues par des simulateurs tels que Genesis. Cependant, les temps de rédaction des programmes de simulation peuvent être comparables.

De plus, la modification d'un code numérique dédié pour tenir compte de nouveaux constituants, nouvelles architectures dans les arborescences dendritiques ou les répartitions de neurones dans les réseaux peut s'avérer impossible alors que Genesis (et d'autres simulateurs du même type) permet de telles transformations de façon relativement aisée et performante.

D'autres avantages peuvent être mis en avant pour de tels simulateurs tels que la possibilité de transporter le code (le programme) sur des ordinateurs dits parallèles où les opérations se font à une plus grande vitesse, ou bien la possibilité de changer aisément de techniques d'intégration numérique ou de créer de nouvelles formes d'illustration graphique des résultats obtenus.

La possibilité d'une sorte de standardisation de la modélisation neuronale peut elle même être utile dans la mesure où elle permet une meilleure communication entre les développeurs dans ce domaine. Ainsi, l'équipe de Caltech a mis en place un système, dénommé BABEL, qui est constitué d'une base de données de programmes de type Genesis qui peut être utilisée par les membres du groupe.

II. Genesis : caractéristiques de base

⊗Genesis a été développé pour réaliser la simulation d'ensembles neuronaux biologiquement réalistes. Trois objectifs ont été poursuivis : le simulateur doit permettre de faire un choix dans le niveau de complexité, depuis le neurone simple isolé à un compartiment jusqu'aux réseaux de neurones à plusieurs compartiments.

Il doit être également ouvert dans son développement propre et offrir la possibilité de communication pour optimiser l'effort de modélisation.

⊗La flexibilé de Genesis, en tant qu'outil de simulation, réside dans le type de programmation utilisé appelé *orienté objet* . Le simulations sont constituées de structures modulaires, sortes de "briques", chacunes de ces briques réalisant une fonction bien précise, chacune de ces briques étant en relation avec les autres briques de façon standardisée. On peut ainsi choisir un niveau de complexité en choisissant les briques adéquates. L'utilisateur peut assembler ces briques à sa guise. De la même façon, l'adjonction de nouveaux modules construits à l'aide de ces briques, peut donner lieu à de nouvelles possibilités de simulation.

⊗L'interface de Genesis est constituée de deux parties. Le niveau le plus bas est le SLI, ou Script Language Interpreter. Ceci est un langage de programmation de type interpréteur, comme celui du système UNIX lui même, avec un nombre important de commandes. L'interpréteur peut lire les programmes SLI (les scripts) soit de façon interactive à partir du clavier, ou bien à partir de fichiers. Il existe une interface fraphique dénommée XODUS (X-windows Output and Display Utility for Simulation). Cette interface est un moyen de haut niveau et très convivial pour faire du développement de programmes et controler leur éxécution. XODUS est constitué de modules graphiques qui participent de la même démarche de programmation des objets neuronaux, sauf qu'ils réalisent des fonctions graphiques.

III. Introduction aux exemples

Ces exemples traiteront de simulations se situant à plusieurs niveaux, du neurone isolé aux petits sytèmes. Ces exemples peuvent donner lieu à des expérimentations numériques sans connaissances particulières du langage. Par le suite, nous verrons comment fondamentalement modifier ces scripts pour réaliser vos propres simulations. Les exemples seront décrits, par chapitre. Dans ce même chapitre, le script *Neuron* (en fait neuron.g) permettra de prendre contact avec le simulateur. Les simulations montreront, pour un modèle neuronal composé d'un soma et de deux dendrites en série, les réponses à diverses sortes d'impulsions électriques, sous le seuil de déclenchement de potentiels daction et au dessus du seuil.

Le chapitre suivant Chap C, est consacré à des expériences et à une modélisation initialement développée par Hodgkin et Huxley (HH) pour comprendre la dépendance des conductances ioniques et leur évolution temporelle, ces conduc-

tances intervenant de manière cruciale dans la génération des potentiels d'action. Dans l'exemple dénommé *Squid* (le calmar !), en référence à l'animal ayant été utilisé par ces auteurs pour mener leurs recherches sur le potentiel d'action, des expériences simulées dites à courant imposé et tension imposée, telles qu'elles sont réalisées in vivo en laboratoire, seront possibles.

Dans le chapitre D, on aborde les propriétés de "cable" des structures axonales et dendritiques constituées de compartiments dits passifs, c'est à dire ne possèdant pas de canaux ioniques actifs, de type (HH). Malgrès cela, de nombreuses informations peuvent être obtenues en faisant appel aux propriétés passives, en particulier en ce qui concerne la morphologie neuronale.

Le chapitre suivant, ChapE, combine des éléments des deux chapitres précédents pour construire un modèle neuronal multicompartimental avec propriétés actives. L'exemple traité est dénommé *Neuron* (déjà introduit). Il introduit les modèles canaux synaptiquement activés. La notion de sommation temporelle provenant de plusieurs entrées synaptiques sera développée. Le système est composé de plusieurs compartiments dendritiques spatialement séparés, contenant des canaux synaptiquement activés de nmanière excitatrice et /ou inhibitrice et un soma possèdant des canaux Sodium et Potassium de type (HH).

Dans le chapitre F, deux simulations seront réalisées qui auront pour but de montrer comment des interactions entre différents types de canaux potentiel dépendants peuvent entrainer des potentiels d'action de structure différente plus complexe (avec des patterns différents) que ceux engendrés par des modèles de type (HH). Ainsi, dans une simulation du comportement d'un soma de cellule "pacemaker" de mollusque, une combinaison particulière de canaux permettra d'engendrer des "bursts" périodiques de potentiels d'action. Dans une seconde simulation, relative à une cellule pyramidale de l'hyppocampe, les bursts résulteront d'une interaction entre le soma et des régions dendritiques lointaines.

Le chapitre G introduit pour la première fois les réseaux de neurones. On considèrera un réseau simple qui représentera un Générateur Central de Patterns (CPG, en anglais). Des circuits de ce type ont été mis en évidence dans les mécanismes de controle d'activités rythmiques, pour un nombre important d'espèces animales. Le modèle de CPG considéré ici est plus schématique et abstrait que le système réel qui pourra cependant produire les mêmes patterns. On s'intéresse au role joué par les interactions entre les quatre neurones dont la structure est identique à celle uitlisée dans le chapitre E.

IV. Introduction à l'interface graphique de Genesis

Avant de commencer l'étude des exemples, il est tout d'abord nécessaire de formuler les bases du fonctionnement de l'interface graphique, XODUS. A l'aide de cette interface, il sera en fait possible de faire un certain nombre de simulations en entrant de nouveaux paramètres sans pour cela intervenir sur les scripts. XODUS permet de visualiser, de plusieurs façons, l'état de plusieurs variables de caractère neurobiologique. Afin de se familiariser avec XODUS, comme cela a été annoncé plus haut, le script *Neuron* va servir de support, et sera décrit plus en détail au chapitre E.

IV.1 Exécution de la simulation.

Les fichiers utiles pour effectuer les simulations se trouvent dans un répertoire particulier dans lequel on pourra avoir accès pour obtenir des informations (textes de programmes, conseils d'utilisation, bibliothèques, etc). Ce dernier se dénomme genesis. (L'exécutable c.a.d le logiciel lui même, porte le même nom, ce qui peut induire en erreur). Pour savoir où se trouve le répertoire genesis (on dira la distribution genesis), on peut éxécuter la commande Linux suivante (compatible Unix) :

find . -name genesis -print

Ceci nous indique la localisation du répertoire genesis et de l'exécutable également. En séance de TD-TP, on copiera un répertoire particulier appelé *neuron* dans son propre répertoire , on lancera genesis et éxécutera le programme *Neuron.g* . Les programmes éxécutables dans genesis portent l'extension .g . Pour cela, apres avoir entré au clavier genesis et s'être assuré d'être dans le répertoire possédant le programme *Neuron.g* (voir TD), on entre au clavier Neuron.g , avec N majuscule (!).

IV.2 Le tableau de contrôle et les boites de dialogues

Il apparait le panneau suivant et d'autres fenêtres sur lesquelles les résultats de la simulation vont se dérouler.

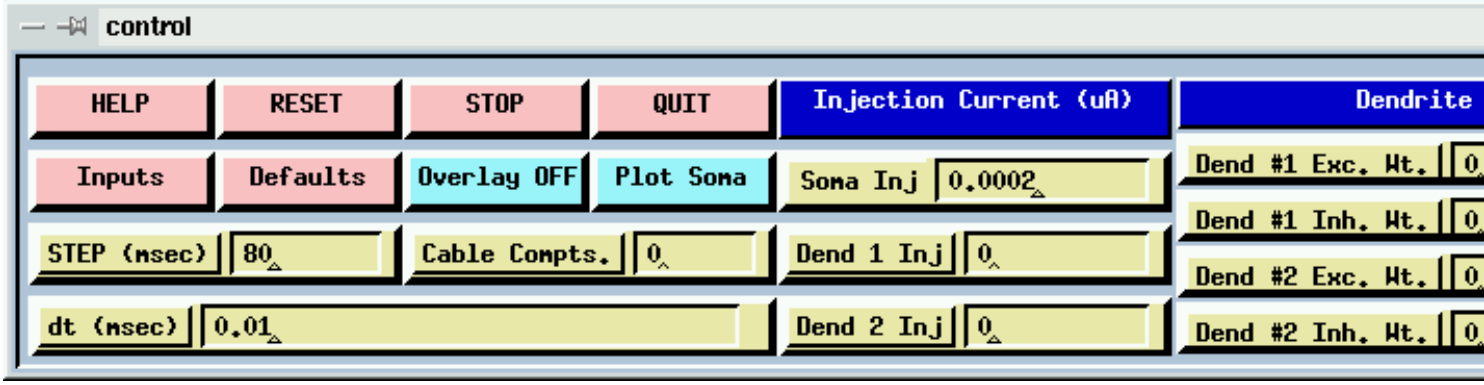


Figure 1

Les fenêtres sont des exemples de *formes (forms)* graphiques créées par genesis. Comme toute les autres fenêtres utilisées par X Windows, qui est le système de fenêtrage général utilisé ici par le système d'exploitation Linux, les fenêtres peuvent être déplacées, redimensionnées à l'aide de la "souris".

Les différents éléments apparaissant sur le panneau s'appellent *bouton (button), label (label), boite de dialogue (dialog), bascules (toggle).* Les boutons *HELP (Aide), RESET (Remise a zéro), STOP et QUIT (quitter)* sont présents dans toutes les simulations. En cliquant sur l'un de ces boutons avec la souris gauche, une action est opérée.

Sous le label (ou intitulé) *Injection Current*, se trouvent 3 boites de dialogue qui permettent de connaitre les quantités de courant injectés dans le neurone considéré ou pour modifier ces valeurs directement du clavier. Le modèle neuronal est ici le suivant :

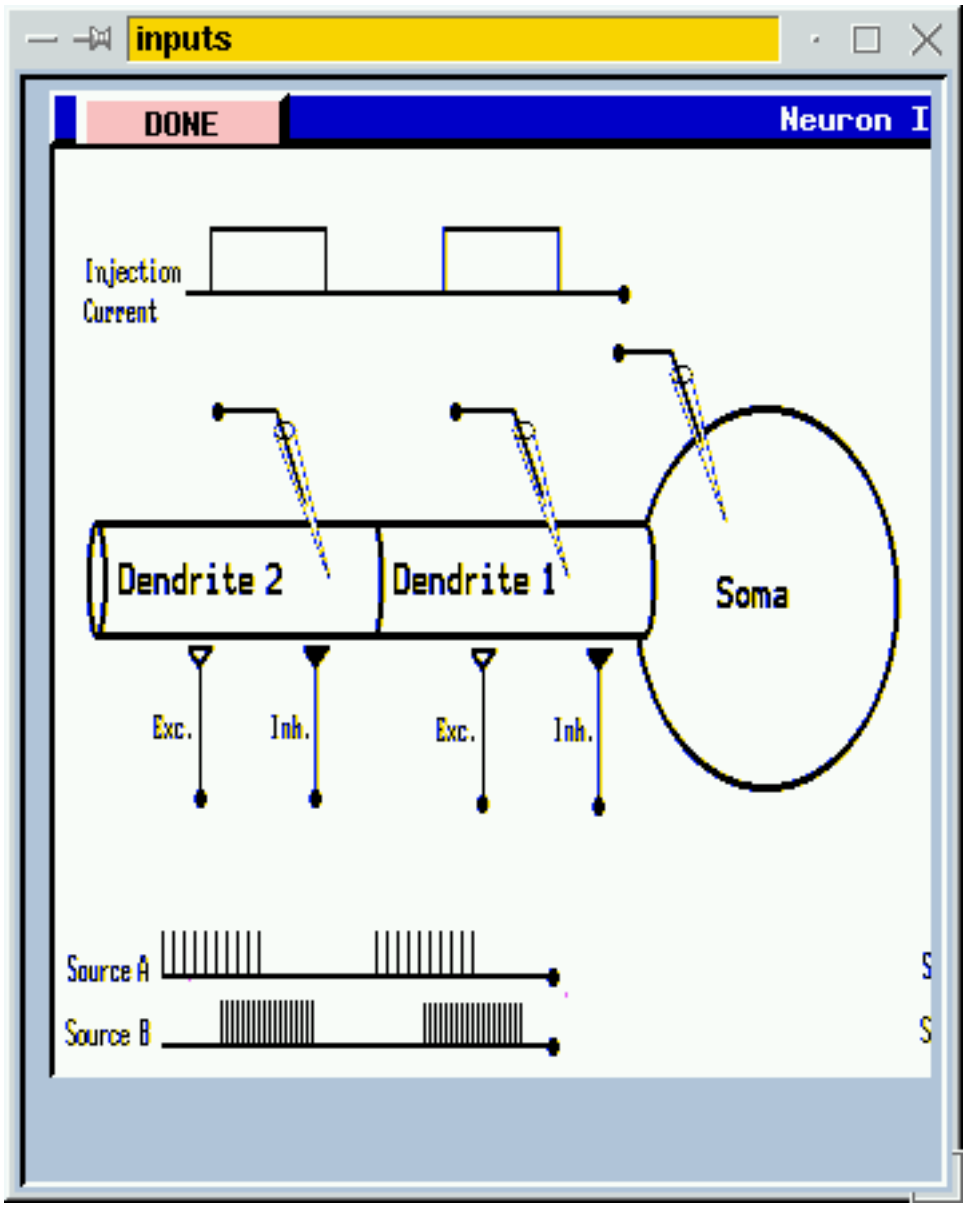


Figure 2

Pour injecter un courant donné dans le soma, on modifie la valeur apparaissant dans le "dialogue" *Soma Inj*. Pour faire cela, on positionne avec la souris le curseur dans la boite correspondante. Ensuite, on utilise les flêches →et ←se trouvant sur le clavier pour déplacer le signe ˆ . Pour supprimer des chiffres, on utilise la touche *Delete* sur le clavier.

Pour enregistrer ces modifications, il est impératif de fonctionner la touche Return (Entrée) au clavier. Ne pas oublier cette opération, cela est souvent une source d'erreur. On peut également enregistrer en cliquant dans la boite contenant le label correspondant (ici sur *Soma Inj*).

Les dialogues intitulés *Overlay OFF* et *Plot Soma* sont des exemples de bascules. Une bascule permet de passer d'un état à un autre. Par exemple, en cliquant sur *Overlay OFF* , on passe à *Overlay ON* qui permet de représenter le graphique suivant dans la même fenêtre que le graphique précédent. On peut ainsi comparer les résultats de deux simulations différentes.

En cliquant sur le bouton *HELP*, on fait apparaitre dans une fenêtre, la figure 4 ci dessus qui illustre le modèle neuronal considéré. De manière générale, dans les exemples suivants (ou tutorials), des informations concernant la simulation sont disponibles dans ces fenêtres.

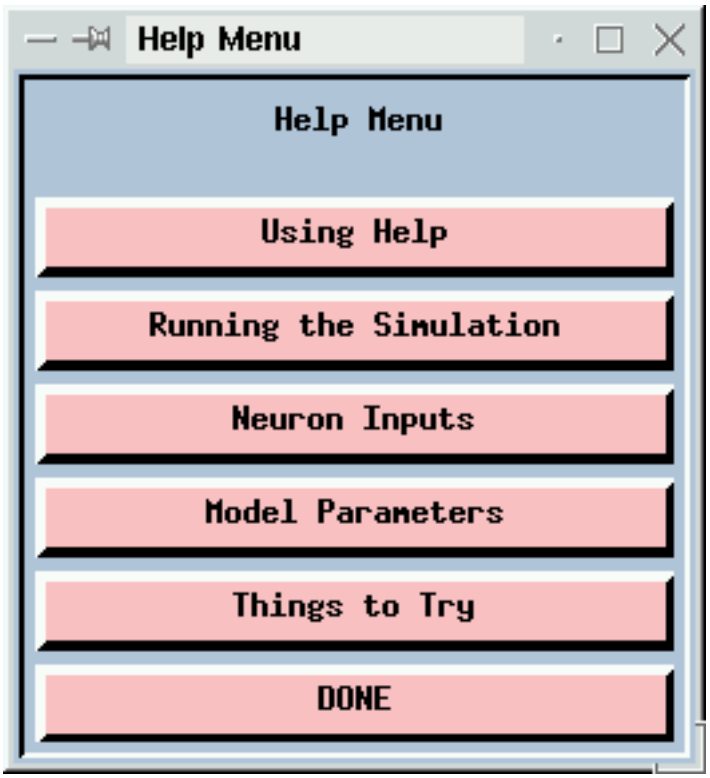


Figure 3

Par exemple, en cliquant sur le bouton *Running the Simulation,* un texte apparait (en anglais) décrivant le modèle (figure 5 ci dessous). L'introduction de ces commentaires, sous cette forme, avec des possibilités de déroulement de texte, est intéressante pour l'utilisateur. Par le suite, dans le développement de programmes par les étudiants(es), cette approche sera encouragée.

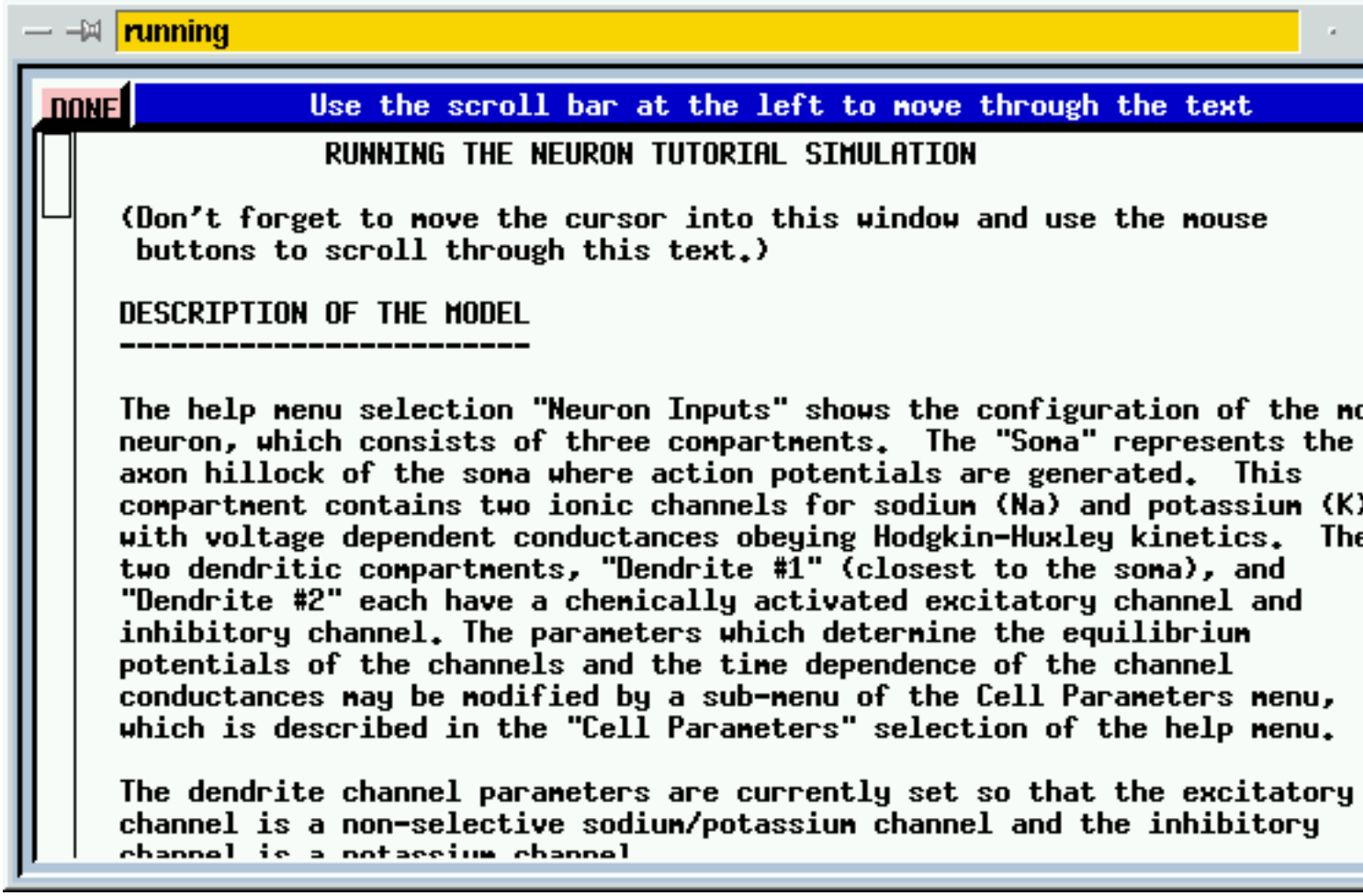


Figure 4

A l'aide du bouton *Models Parameters*, le schéma électrique de l'un des compartiments du système est indiqué (figure 5).

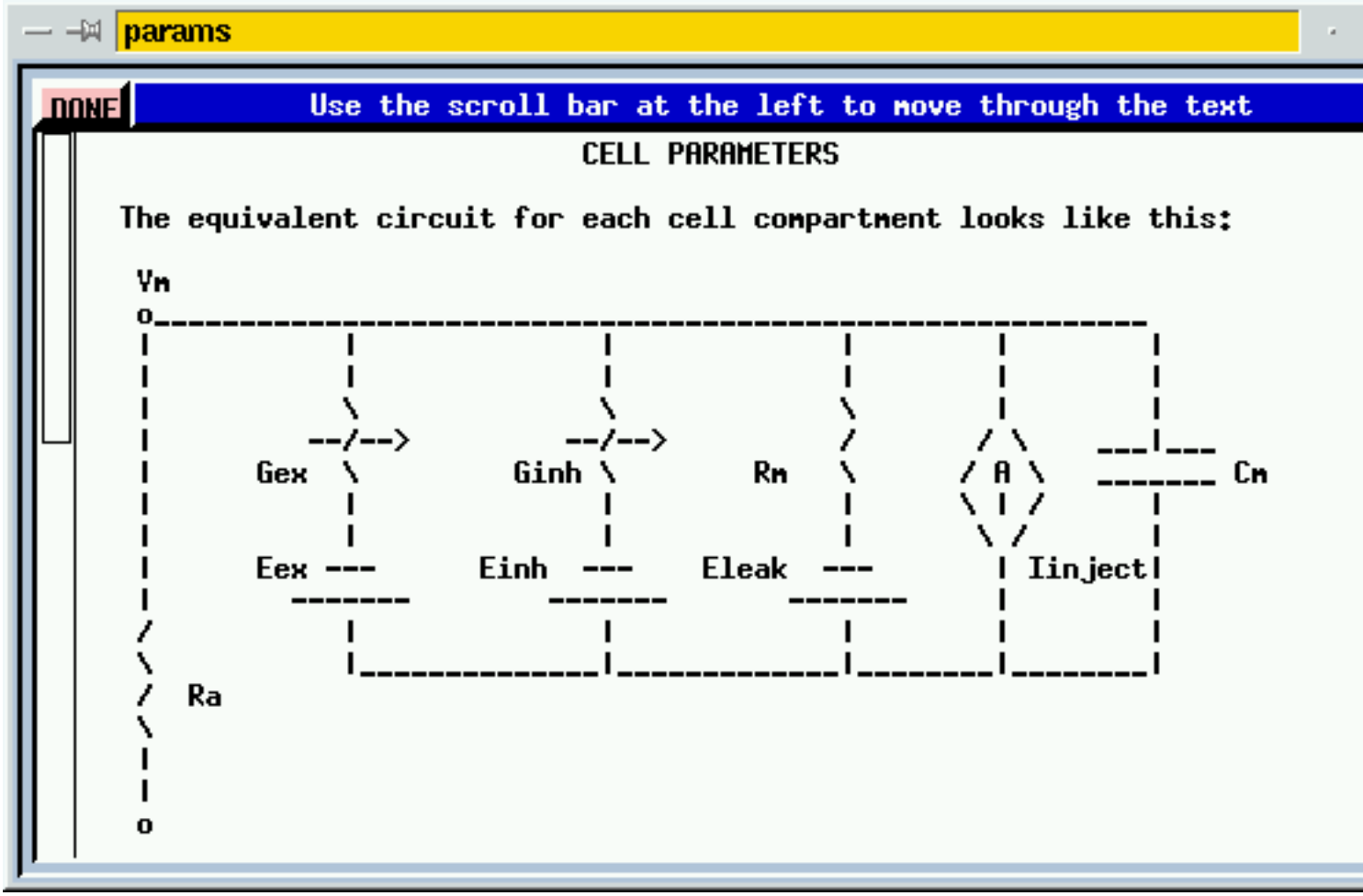


Figure 5

En fait, le modèle neuronal considéré ici (figure 2), est composé d'un compartiment soma et deux compartiments dendritiques qui sont connectés par des résistances axiales (les résistances Ra sur la figure 5). Le soma possède des canaux potentiels dépendants du type Hodgkin Huxley (voir plus loin). Les compartiments dendritiques, ou dendrites, possèdent des canaux activés synaptiquement qui répondent à des potentiels d'action (ou spikes) appliqués aux synapses. Ces trains de spikes représentent des entrées possibles délivrées à partir d'axones d'autres neurones. Cette simulation *Neuron.g* sera reprise par la suite pour étudier les propriétés de ces canaux synaptiques.

Sur la figure 2, on peut voir que l'on peut injecter des pulses de courant (*Injection Current*) dans l'un quelconque ou plusieurs compartiments. On peut également connecter des trains de spikes à l'une ou plusieurs synapses avec une amplitude spécifique (ou poids) sur chaque synapse. Ces paramètres (choix des synapses, amplitude, courants injectés) sont modifiables à l'aide du bouton *Inputs*

du tableau de contrôle.

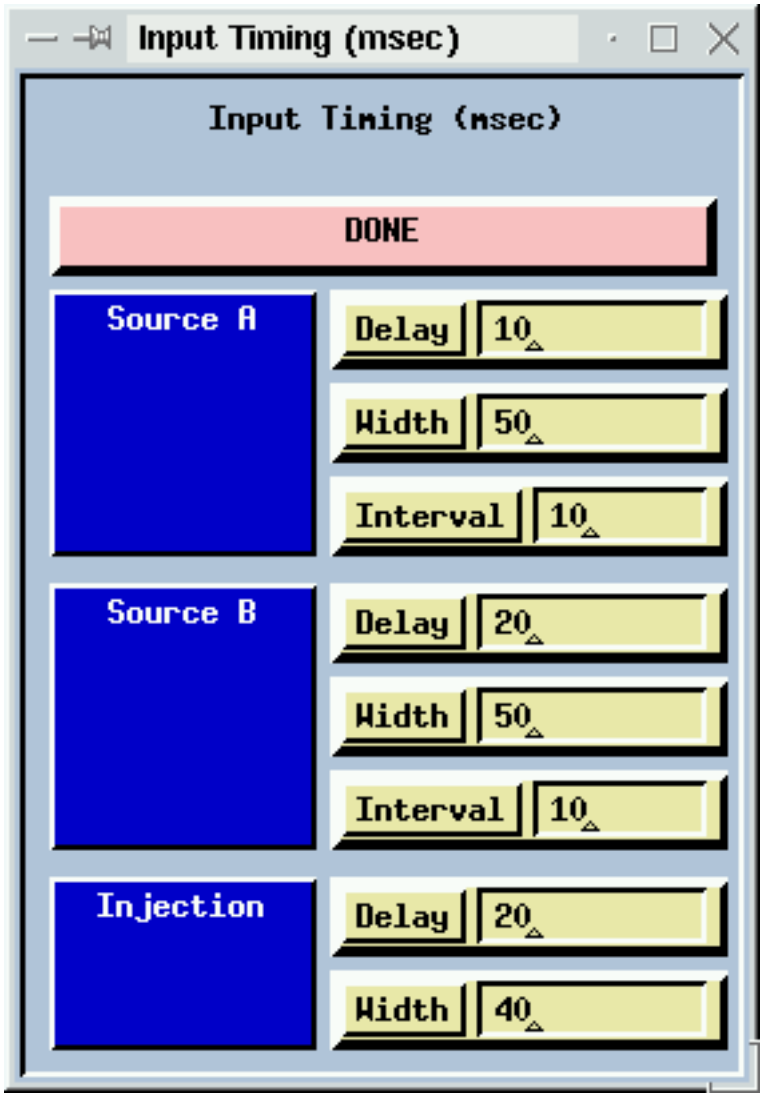


Figure 6

Le bouton *Cable Compts*, dans le panneau de controle (control panel), il est possible de modifier le nombre de compartiments "passifs" (c.a.d sans canaux potentiel dépendants, pas de conductance ionique variable) éventuellement présents entre les deux compartiments dendritiques. Ceci permet de mesurer l'effet d'entrées sur le neurone qui sont spatialement séparées.

IV.3 Visualisation des résultats

. En utilisant les valeurs par défaut des paramètres, on obtient les figures suivantes, par lancement de la simulation à l'aide du bouton *STEP* :

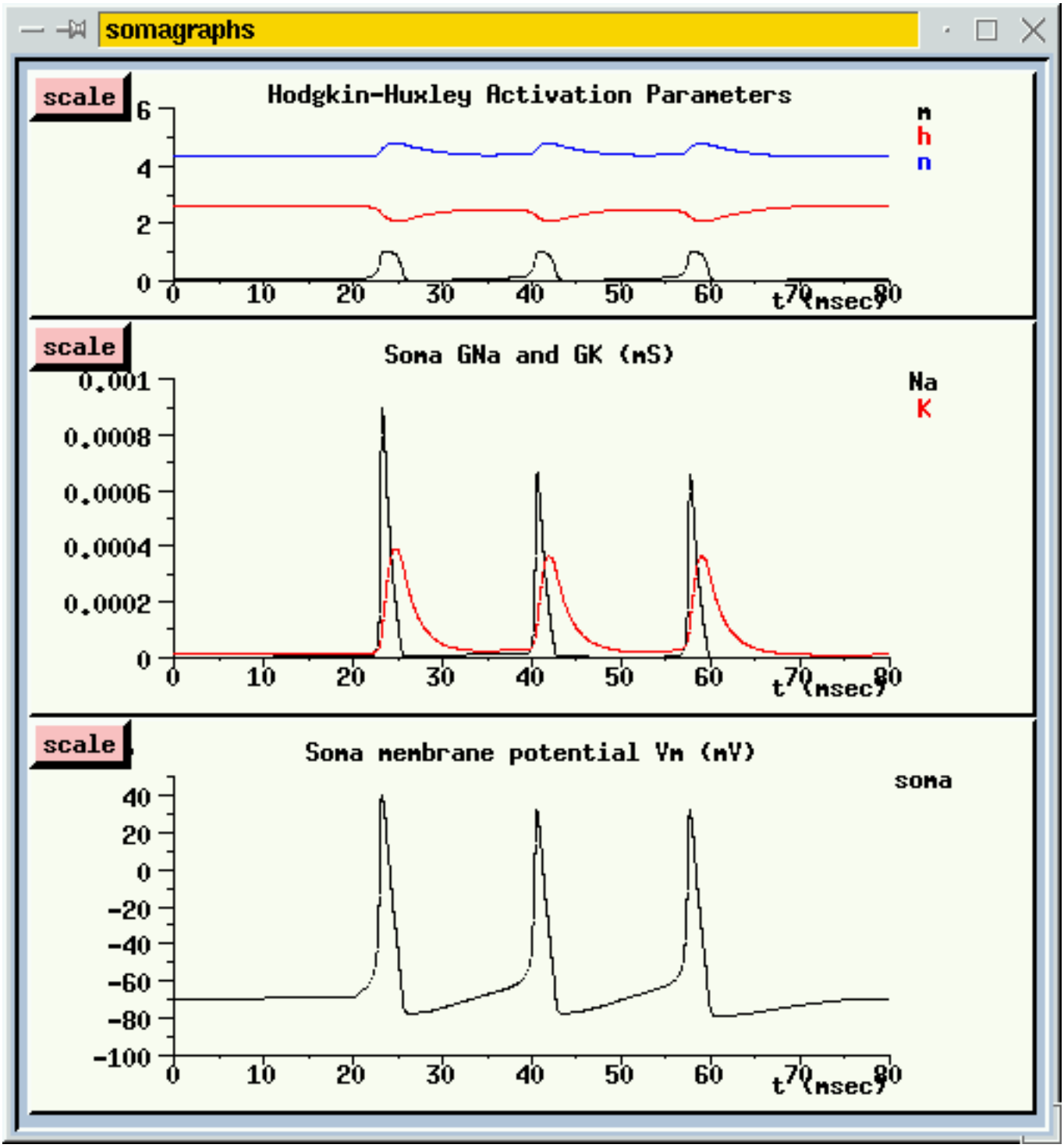


Figure 7

Ne pas oublier de faire *RESET* avant cette opération. Ceci efface tous les graphiques présents et réinitialise le processus. La valeur indiquée dans le dialogue

STEP indique le temps maximum de la simulation (ici 80 ms), qui s'effectue avec des pas de temps de $0.01ms$. En fait, il est possible dans Genesis de visualiser toutes les variables du système. Par exemple, sur la figure 7, sont représentées les variations de potentiel membranaire V_m au soma montrant des potentiels d'action produits et les variations de conductances Sodium et Potassium correspondantes. Les paramètres de Hodgkin Huxley de ces conductances sont également visualisés. Ces paramètres seront introduits plus loin.

L'interface XODUS permet d'avoir un controle souple des graphes (ou graphiques) qu'elle produit. Par exemple, les échelles de ces graphes peuvent être modifiées en cliquant sur sur le bouton *SCALE* , puis en entrant la valeur souhaitée et en pressant le bouton *DONE*.

V. Les éléments de base du langage Genesis

1 : tutorial1.g

Dans un premier temps, on copie le script intitulé tutorial1.g, se trouvant dans ResNeur, dans son répertoire de travail.

Ce script guide l'utilisateur dans une brève introduction à Genesis, en utilisant les éléments de base du simulateur pour créer et éxécuter une simulation simple d'un compartiment passif.

On copie le script dans son propre répertoire. En cliquant sur l'icone intitulée Home se trouvant sur le "bureau", on peut effectuer cette copie.

On peut visualiser le texte de tutorial1.g par la commande *more* que l'on tape après le "prompt" $ présent dans une fenêtre de travail (ou console). On peut également lancer le progiciel Genesis en tapant \genesis et ensuite utiliser la même commande *more*.

genesis #33 > more tutorial1.g (affichage du texte du programme)

//genesis script pour une simulation simple d'un neurone composé d'un seul compartiment passif (pas de canaux ioniques, stimulation faible par courants injectés)

Les mots réservés, non modifiables par l'utilisateur, sont de couleur différente.

// : ce symbole signifie que le texte qui le suit est un commentaire, qui ne sera pas réalisé lors de l'éxécution.

// création d'un élémént parent. "neutral" est un *objet* du langage Genesis
// et "cell" est un nom pour cet objet fourni par l'utilisateur. Le symbole /
//signifie que cet objet se trouve à la racine.
create neutral /cell

// création d'un objet "compartment" pour cette cellule dénommée "cell". On crée un compartiment somatique.
create compartment /cell/soma
Cet objet compartimental possède certaines caractéristiques, définies dans des "champs " ("fields"). On peut avoir accès à ces champs par l'ordre showfield (voir plus loin). On affecte des valeurs à ces champs par l'ordre "setfield". Il a été dénommé /cell/soma par l'utilisateur .
// set some internal fields
setfield /cell/soma Rm 10 Cm 2 Em 25 inject 5

// Création et affichage d'un graphique dans une "forme" (objet graphique). /data et /data/voltage peuvent avoir leur nom modifié par l'utilisateur.
create xform /data
create xgraph /data/voltage
xshow /data

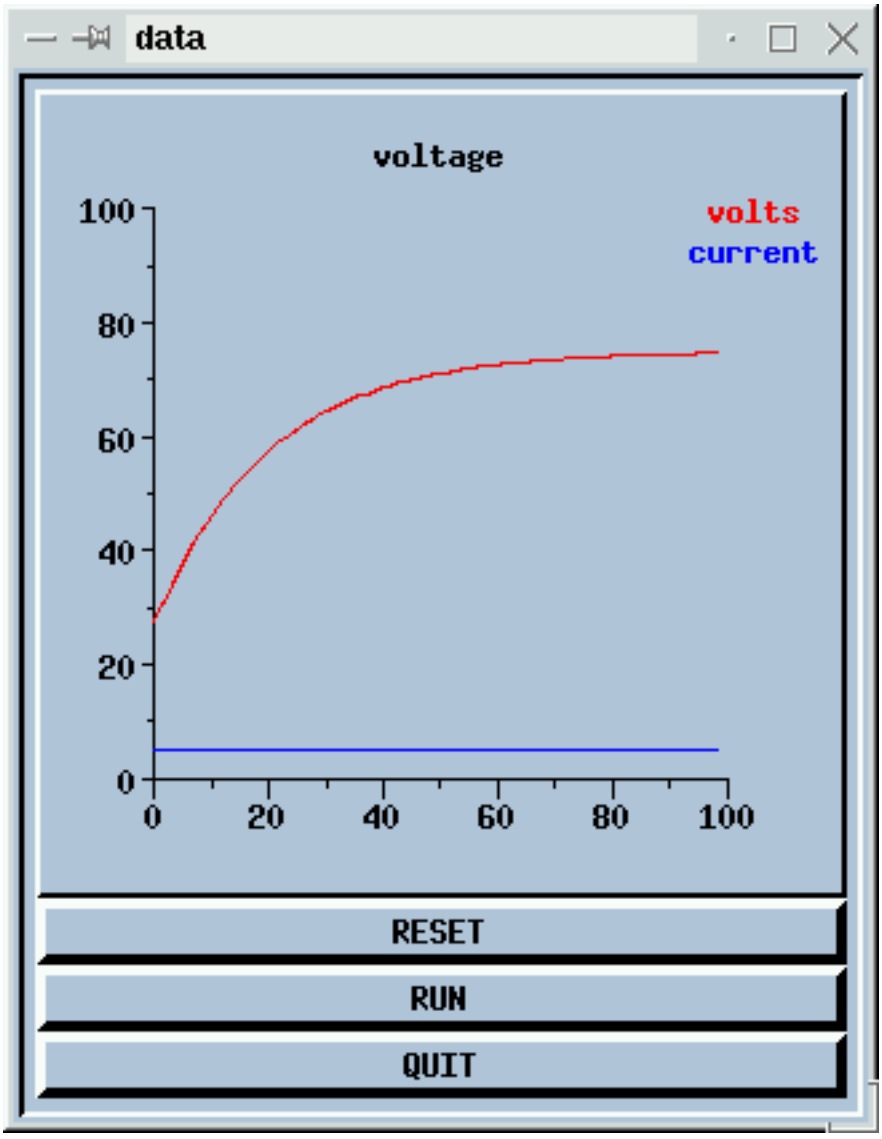
// On envoie un "message" depuis l'objet somatique (/cell/soma) vers l'objet graphique /data/voltage. Le titre du graphe est voltage. On représente dans l'exemple qui suit les variations dans le temps du potentiel membranaire de l'objet /cell/soma quand un courant de 5 pA est injecté dans ce soma. En effet, parmi les champs accessibles figure ce potentiel (voir plus loin avec showfield) dénommé Vm. *volts signifie que la courbe représentant Vm sera appelée Volts. Elle sera de couleur rouge. De la même façon, la courbe repésentant le courant injecté (ici une constante) sera appelée current. Elle sera de couleur bleue.
addmsg /cell/soma /data/voltage PLOT Vm *volts *red
addmsg /cell/soma /data/voltage PLOT inject *current *blue

// make some buttons to execute simulation commands
// On crée des boutons pour éxécuter certaines commandes Genesis. Le bouton RESET (mot défini par l'utilisateur) s'éxécute à l'aide de l'ordre "reset" . La syntaxe est celle indiquée, à l'aide du mot script, précédé de -. Reset signifie l'initialisation du processus. Cet ordre reset doit être éxécuté à chaque ouverture de la

fenêtre graphique. De même, l'ordre RUN (mot défini par l'utilisateur) s'éxécute par le script "step 100", qui signifie que la simulation s'effectue sur 100 pas de temps, la longueur de ces pas de temps pouvant être modifiée. Avec le script quit, on quitte Genesis.

```
create xbutton /data/RESET -script reset
create xbutton /data/RUN -script "step 100"
create xbutton /data/QUIT -script quit

check       // On effectue un contrôle de la consistence des ordres envoyés.
reset       // On initialise chaque élément avant éxécution.
// Fin du texte du programme
```



Dans ce qui suit, on montre comment avoir accès aux divers champs des objets construits dans le programme précédent (on peut dire aussi le script précédent).

genesis #34 > showfield /cell/soma -all

Ceci est la réponse :

[/cell/soma]
x0y0z0 = (0.000000e+00 , 0.000000e+00 , 0.000000e+00)
xyz = (0.000000e+00 , 0.000000e+00 , 0.000000e+00)
flags = 80
FUNCTIONAL
Clock [0] = 1.000000e+00
0 incoming messages
2 outgoing messages
————————————————————-
activation = 0
Vm = 74.66310265
previous_state = 74.64582955
Im = 0
Em = 25
Rm = 10
Cm = 2
Ra = 0
inject = 5
dia = 0
len = 0
initVm = 25

La valeur de activation=0 signifie qu'aucun spike a été émis. La valeur de Vm indiquée est celle correspondant au dernier pas de temps de la simulation. Previous_state correspond à la valeur de Vm, 1 pas de temps avant la fin. La valeur de Im correspond au courant transmembranaire, à la fin de la simulation. La valeur du courant injecté est de 5 pA. Rm et Cm sont les résistances membranaires et capacitances transmembranaires de ce compartiment, considéré comme ponctuel (Ra, résistance longitudinale nulle, de même que dia (diamètre) et len (longueur)). Em est la fem d'équilibre et Vm a pour valeur initiale cette valeur Em. En fait, l'équation de la membrane est :

$C_m \frac{dV_m}{dt} = \frac{(-V_m + E_m)}{R_m}$

genesis #35 >

On peut modifier, dans le cours de la simulation, “en ligne”, des valeurs des paramètres entrant dans la définition des caractéristiques du soma, par setfield .. entré directement au clavier, sans modifier le texte du programme. Ceci permet de faire des tests, avant d'éventuelles modifications. Faire reset avant de faire RUN.

genesis #43 > setfield /cell/soma Em 30 inject 6 Rm 12 Cm 3
OK

genesis #44 > showfield /cell/soma -all
..
activation = 0
Vm = 97.52329027
previous_state = 97.39719399
Im = 0
Em = 30
Rm = 12
Cm = 3
Ra = 0
inject = 6
dia = 0
len = 0
initVm = 30

D'autres manipulations par des ordres éxécutés depuis le clavier.

genesis #56 > reset ; step 200 (execution de 200 pas a partir de t=0 et affichage)
time = 200.000000 ; step = 200
completed 200 steps in 0.000000 cpu seconds
genesis #57 > step 100 (execution de 100 pas a partir de t=200 et affichage)
time = 300.000000 ; step = 300
completed 100 steps in 0.000000 cpu seconds
genesis #59 >

2 : Exercice

Lors de l'éxécution de la commande QUIT, on quitte Genesis. Afin de pouvoir éxécuter un autre script sans sortir de Genesis, on va remplacer la commande :

“create xbutton /data/QUIT -script quit” par “create xbutton /data/QUIT -script “hidegraphics / ” et sauver le nouveau script, dans son répertoire de travail, avec un nouveau nom, par exemple pass1.g

La commande “more” considérée plus haut permet d’afficher le texte du script sans pouvoir le modifier. Pour pouvoir modifier le texte, il faut l’éditer. Il est necessaire d’activer un éditeur de texte existant sur le disque (par exemple kedit).

Afin de faciliter ce qui suit, il est préférable d’utiliser les possibilités offertes par le dispositif multiécrans. On active donc un nouvel écran (dans la barre inférieure des menus), on ouvre une fenêtre de commandes d’ordres Linux, on se positionne dans son répertoire et on lance kedit en tapant simplement >kedit&.

Ici, > désigne le “prompt” qui est créé automatiquement. On ouvre alors (ordre open, dans le menu file) le fichier que l’on désire éditer, ici tutorial1.g. On peut faire également >kedit tutorial1.g&. Le signe “&” permet de lancer le logiciel kedit et de “reprendre la main “ pour éxécuter, éventuellement une autre commande.

La modification du texte de tutorial1.g (pour donner pass1.g) porte sur l’introduction de l’ordre (du langage Genesis) dénommé “hidegraphics”. Cet ordre est en fait un script particulier qui a été inséré dans une bibliothèque dénomméé “bib1.g” qui se trouve dans le répertoire ResNeur. Il est necessaire de faire une copie de cette bibliothèque dans son propre répertoire.

Il faudra ensuite “s’attacher” cette bibliothèque en inscrivant, en tête de son propre programme, l’expression suivante : include bib1.g