Who should read this book

This book is about how to use the NEURON simulation environment to construct and apply empirically-based models of neurons and neural networks. It is written primarily for neuroscience investigators, teachers, and students, but readers with a background in the physical sciences or mathematics who have some knowledge about brain cells and circuits and are interested in computational modeling will also find it helpful. The emphasis is on the most productive use of NEURON as a means for testing hypotheses that are founded on experimental observations, and for exploring ideas that may lead to the design of new experiments. Therefore the book uses a problem-solving approach, with many working examples that readers can try for themselves.

What this book is, and is not, about

Formulating a conceptual model is an attempt to capture the essential features that underlie some particular function. This necessarily involves simplification and abstraction of real-world complexities. Even so, one may not necessarily understand all implications of the conceptual model. To evaluate a conceptual model it is often necessary to devise a hypothesis or test in which the behavior of the model is compared against a prediction. Computational models are useful for performing such tests. The conceptual model and the hypothesis should determine what is included in a computational model and what is left out. This book is not about how to come up with conceptual models or hypotheses, but instead focuses on how to use NEURON to create and use computational models as a means for evaluating conceptual models.

What to read, and why

The first chapter conveys a basic idea of NEURON’s primary domain of application by guiding the reader through the construction and use of a model neuron. This exercise is based entirely on NEURON’s GUI, and requires no programming ability or prior experience with NEURON whatsoever.

The second chapter considers the role of computational modeling in neuroscience research from a general perspective. Chapters 3 and 4 focus on aspects of applied mathematics and numerical methods that are particularly relevant to computational neuroscience. Chapter 5 discusses the concepts and strategies that are used in NEURON to simplify the task of representing neurons, which (at least at the level of synapses and cells) are distributed and continuous in space and time, in a digital computer, where
neither time nor numeric values are continuous. Chapter 6 returns to the topic of model construction, emphasizing the use of programming.

Chapters 7 and 8 provide "inside information" about NEURON’s standard run and initialization systems, so that readers can make best use of their features and customize them to meet special modeling needs. Chapter 9 shows how to use the NMODL programming language to add new biophysical mechanisms to NEURON. This theme continues in Chapter 10, which starts with mechanisms of communication between cells (gap junctions, graded and spike–triggered synaptic transmission), and moves on to models of artificial spiking neurons (e.g. integrate and fire cells). The first half of Chapter 11 is a tutorial on NEURON’s GUI tools for creating simple network models, and the second half shows how to use the strengths of the GUI and hoc programming to create more complex networks.

Chapter 12 discusses the elementary features of the hoc programming language itself. Chapter 13 describes the object–oriented extensions that have been added to hoc. These extensions have greatly facilitated construction of NEURON’s GUI tools, and they can also be very helpful in many other complex programming tasks such as creating and managing network models. Chapter 14 presents an example of how to use object oriented programming to increase the functionality of NEURON.

Table of contents

Note: page numbers in each chapter start from 1.

Chapter 1. A tour of the NEURON simulation environment

Modeling and understanding 1
Introducing NEURON 1
1. State the question 2
2. Formulate a conceptual model 2
3. Implement the model in NEURON 4
   Start NEURON and bring up a CellBuilder 5
   Enter the specifications of the model cell 6
      Topology 6
      Subsets 8
      Geometry 10
      Biophysics 12
   Save the model cell 14
   Execute the model specification 16
4. Instrument the model 17
Chapter 2. Principles of neural modeling

Why model? 1
From physical system to computational model 1
  Conceptual model: a simplified representation of a physical system 1
  Computational model: an accurate representation of a conceptual model 2
  An example 3

Chapter 3. Expressing conceptual models in mathematical terms

Chemical reactions 1
  Flux and conservation in kinetic schemes 2
  Stoichiometry, flux, and mole equivalents 3
  Compartment size 5
    Scale factors 7
Electrical circuits 8
Cables 14

Chapter 4. Essentials of numerical methods for neural modeling

Spatial and temporal error in discretized cable equations 1
  Analytic solutions: continuous in time and space 2
  Spatial discretization 4
  Adding temporal discretization 6
Numerical integration methods 7
  Forward Euler: simple, inaccurate and unstable 7
    Numerical instability 9
  Backward Euler: inaccurate but stable 11
Crank–Nicholson: stable and more accurate 12
Efficient handling of nonlinearity 14
Adaptive integration: fast or accurate, occasionally both 16
Implementational considerations 17
The user’s perspective 18
Error control 23
Local variable time step method 23
Discrete event simulations 25
Error 26
Summary of NEURON’s integration methods 28
Fixed time step integrators 28
Default: backward Euler 28
Crank–Nicholson 29
Adaptive integrators 29
CVODE 30
DASPK 30

Chapter 5. Representing neurons with a digital computer

Discretization 1
How NEURON separates anatomy and biophysics from purely numerical issues 2
Sections and section variables 3
Range and range variables 4
Segments 5
Implications and applications of this strategy 6
Spatial accuracy 6
A practical test of spatial accuracy 7
How to specify model properties 8
Which section do we mean? 8
1. Dot notation 8
2. Stack of sections 9
3. Default section 9
How to set up model topology 10
No loops of sections 10
Chapter 6. How to build and use models of individual cells

GUI vs. hoc code: which to use, and when? 1
Hidden secrets of the GUI 2
Implementing a model with hoc 2
  Topology 3
  Geometry 5
  Biophysics 5
Testing the model implementation 5
  An aside: how does our model implementation in hoc compare with the output of the CellBuilder? 7
Instrumenting a model with hoc 10
Chapter 7. How to control simulations

Simulation control with the GUI
The standard run system
An outline of the standard run system
fadvance()
advance()
step()
steprun() and continu-run()
run()
Details of fadvance() (The fixed step methods: implicit Euler and Crank–Nicholson
Adaptive integrators and discrete events
Incorporating graphs and new objects into the plotting system

Chapter 8. How to initialize simulations

State variables and STATEs
Basic initialization in NEURON: finitalize()
Default initialization in the standard run library: stdinit() and init()
INITIAL blocks in NMODL
Default initialization of STATEs
Ion concentrations and equilibrium potentials 7
Initializing concentrations in hoc 10
Examples of custom initializations 11
  Initializing to a particular "resting potential" 11
  Initializing to steady state 13
  Initializing to a desired state 14
  Initializing by changing the model 14
  Details of the mechanism 15
  Initializing the mechanism 17

Chapter 9. How to expand NEURON’s library of mechanisms
  Overview of NMODL 1
  Example 9.1: a passive "leak" current 2
  Example 9.2: a localized shunt 7
  Example 9.3: an intracellular stimulating electrode 10
  Example 9.4: a voltage–gated current 12
  Example 9.5: a calcium–activated voltage–gated current 19
  Example 9.6: extracellular potassium accumulation 24
  General comments about kinetic schemes 28
  Example 9.7: kinetic scheme for a voltage–gated current 30
  Example 9.8: calcium diffusion with buffering 35
  Example 9.9: a calcium pump 44
  Models with discontinuities 48
    Discontinuities in PARAMETERs 48
      Time dependent PARAMETER changes 49
    Discontinuities in STATEs 50

Chapter 10. Synaptic transmission and artificial spiking cells
  Modeling communication between cells 1
    Example 10.1: graded synaptic transmission 2
    Example 10.2: a gap junction 5
    Modeling spike–triggered synaptic transmission: an event–based strategy 7
      Conceptual model 7
Chapter 11. Modeling networks

*Note: this chapter is in an early draft, so we present its outline.*

**Building a simple network with the GUI**

- Conceptual model of recurrent inhibition:
  - Motoneuron with excitatory afferent and Renshaw cell
- Design of the computational model:
  - Represent the motoneuron with a biophysical model
  - Represent the afferent spike train and Renshaw cell with artificial cell models
  - Use the event delivery system to represent axonal conduction delays and synaptic latency
- Implementing the computational model
  - Specifying the motoneuron model: the NetReadyCellGUI tool
  - Specifying the afferent spike train and the Renshaw cell: the ArtCellGUI tool
  - Specifying the network connections and creating the net: the NetGUI tool (Network Builder)
- Running a simulation and plotting spike trains

**Combining the GUI and programming to build a complex network model**

- Conceptual model of inhibitory synchronization
  - The cells: spontaneously spiking neurons with a range of natural firing frequencies
  - Network architecture: fully connected inhibitory net
Design of the computational model

Represent the cells with artificial spiking neurons
Use the event delivery system to represent axonal conduction delays and synaptic latency

Implementing the computational model

Using the ArtCellGUI tool to create a hoc file that defines the basic cell class
Using the NetGUI tool to create a hoc file that contains the basic procedures needed to manage connectivity and display results
Exploiting the GUI-generated code by writing a handful of procedures that spawn cell instances, set up the connections between them, and display simulation results

Chapter 12. hoc -- NEURON's interpreter

The interpreter in $NEURONHOME/bin/$CPU
Constructing programs that contain the interpreter
The stand-alone interpreter
  Starting the interpreter
  Error handling
Syntax
  Names
  Variables
  Expressions
  Statements
  Comments
Flow control
Functions and procedures
  Arguments
    Call by value vs. call by reference
    Local variables
    Recursive functions
Input and output
Editing
Chapter 13. Object–oriented programming

Object vs. class 1
The object model in hoc 1
Objects and object references 2
  Declaring an object reference 2
  Creating and destroying an object 2
  Using an object reference 3
  Defining an object template 3
    Direct commands 4
    Initializing variables in an object 4
    Keyword names 5
Object references vs. object names 5
  An example of the didactic use of object names 6
Using objects to solve programming problems 7
  Dealing with collections or sets 7
    Arrays 7
      Example: emulating an array of strings 7
    Lists 8
      Example: a stack of objects 8
    Encapsulating code 9
Polymorphism and inheritance 10

Chapter 14. How to modify NEURON itself

Graphical interface programming 1
  General issues 2
    A pattern for defining a template 3
    Enclosing the GUI tool in a single window 4
    Saving the window to a session 6
  Tool–specific development 9
    Plotting 9
    Handling events 13
Finishing up 15