Übungen zur Vorlesung

# Mathematische Aspekte der Modellierung und Simulation in den Neurowissenschaften

Dr. S. Lang, D. Popović

Abgabe: 15. Juni 2010 in der Übung

---

## Übung 18 Einführung in neuroDUNE (2 Punkte)

In dieser Aufgabe wird neuroDUNE, eine Umgebung zur numerischen Simulation räumlich verteilter Neuronen, vorgestellt. Die Software läuft unter Linux und soll im Pool installiert werden. Sie baut auf dem Numerik-Framework *Dune* (www.dune-project.org), Version 1.2.2, auf und ist im Pool getestet. Sie können gerne auch privat installieren, allerdings kann dann keine Installations-Hilfe gegeben werden.

1. Im Pool finden Sie unter dem Verzeichnis `/export/home/dune/neuroDUNE` die Software für dieses Übungsblatt. Kopieren Sie von dort die Tarballs `dune-common-1.2.2.tar.gz, dune-grid-1.2.2.tar.gz, dune-istl-1.2.2.tar.gz` und `neurodune-uebung.tar.gz` in einen Ordner in Ihrem Home-Verzeichnis.
2. Entpacken Sie jeden Tarball `<name>.tar.gz` mit dem Befehl

   ```
   tar xfvz <name>.tar.gz
   ```

3. Nach dem Entpacken kopieren Sie die Datei `config.opts` aus dem obigen Ordner in Ihren lokalen Ordner und editieren sie. Sie müssen für die Option `--prefix=...` den Pfad eingeben, unter dem Sie die Tarballs gespeichert haben. Orientieren Sie sich am Beispiel, das für den User `dune` in dessen Home-Verzeichnis im Ordner `neuroDUNE` installiert. Sie können und sollen natürlich gerne einen anderen Pfad verwenden.
4. Führen Sie nun den Befehl

   ```
   ./dune-common-1.2.2/bin/dunecontrol --opts=config.opts all
   ```

   in der Konsole im Ordner, in dem Sie installiert haben, aus. Dies startet das Installations-Skript für Dune und neuroDUNE.
5. Sollten Sie privat installieren, beachten Sie, daß Sie die `autotools, make` und den `gcc` benötigen.
6. Wenn alles geklappt hat, gehen Sie in den Ordner `neuroDUNE-uebung` und sehen Sie sich um. Sie sollten zwei Unterordner vorfinden, `neuroDUNE` und `samples`. Der erste stellt benötigte Klassen und Werkzeuge bereit, der zweite enthält als Beispiel-Applikation das Izhikevich-Modell.
7. Im Ordner des Izhikevich-Modells geben Sie `make clean` und `make` ein, um die Applikation erneut probeweise zu kompilieren. Dann können Sie das Programm mit `./run_izhikevich` ausführen.

Sollte etwas nicht klappen oder Fehler beim Kompilieren auftreten, wenden Sie sich per Mail an den Übungsgruppenleiter oder an die Liste.

## Übung 19 Punktneuronen-Modelle mit neuroDUNE (13 Punkte)

In dieser Aufgabe werden wir mit neuroDUNE einfache Punktneuronenmodelle, die in der Vorlesung vorgestellt wurden, implementieren. Jedes Abgabe-Team erhält ein anderes Modell zur Bearbeitung.

Im `neuroDUNE/cell`-Ordner gibt es eine Datei `cell_base.hh`. Diese stellt eine Basisklasse `CellBase<T,dim>` für alle Neuronenzellen zur Verfügung. Der erste Template-Parameter ist der Typ für das Potential (z.B. `double`), der zweite die Dimension der Zelle (in der Regel 1). Das Punktneuronen-Modell im Izhikevich-Ordner ist von dieser Klasse abgeleitet, jedoch sind nicht alle Methoden implementiert. Die wichtigsten Methoden der Basis-Zelle sind:

```
// base class interface
void TDiscStep(double t, double dt); // Zeitschritt ausfuehren
void UpdateSolution();               // Loesung nach Schritt updaten
void writeData (double t, int step); // Ausgabe des Potentials
```

Als Zeitschrittschema verwenden wir im Moment nur das bekannte Forward Euler Verfahren.

Ihre Aufgabe ist nun:

1. Machen Sie sich mit der Implementierung des Izhikevich-Modells vertraut, starten Sie es und vergleichen Sie mit der früheren Octave-Version, ob die Ausgabe stimmt (keine Abgabe dieses Teils nötig, er dient nur zum kennenlernen!).
2. Suchen Sie sich unten das von Ihnen zu implementierende Modell heraus und eventuell die in der Vorlesung angegebenen Paper, falls die Parameter nicht klar sind.
3. Implementieren Sie das Modell, indem Sie sich am Izhikevich-Modell orientieren. Verwenden Sie am Besten die gleiche Struktur, d.h. legen Sie Sie zwei neue Dateien an: Eine `<meineappl>.hh`-Datei enthält die von der Basis-Klasse abgeleitete Klasse für Ihr Modell und eine weitere `run_<meineappl>.cc`-Datei den Treiber dazu.
4. Um Ihr Modell zu kompilieren, editieren Sie im `pointneuron`-Verzeichnis die Datei `Makefile.am`, in dem Sie ein Make-Ziel für Ihre Applikation hinzufügen. Sie müssen ungefähr folgende Zeilen verändern oder einfügen:

   ```
   ...
   # programs to be built
   noinst_PROGRAMS = run_izhikevich run_meineappl

   # source files of the programs
   run_izhikevich_SOURCES  = run_izhikevich.cc
   run_meineappl_SOURCES   = run_meineappl.cc

   # path to include files
   run_izhikevich_INCLUDES = -I../../neuroDUNE/cell
   run_meineappl_INCLUDES  = -I../../neuroDUNE/cell

   # FLAGS
   run_izhikevich_CXXFLAGS = -DWITH_INDEX_SETS $(run_izhikevich_INCLUDES)
   run_meineappl_CXXFLAGS  = -DWITH_INDEX_SETS $(run_meineappl_INCLUDES)
   ...
   ```

   Im Wesentlichen müssen Sie also einfach alles, was es für die Beispiel-Applikation gibt, für Ihre Applikation duplizieren und anpassen.
5. Führen Sie nun `make clean` und `make` aus, um Ihre Applikation zu kompilieren. Dann können Sie das Programm mit `./run_<meineappl>` ausführen.
6. Die Paramter Ihres Modells sowie die Schrittweite und Simulationsdauer sollten variabel sein. Am Besten wäre es, wenn Sie die Parameter Ihres Modells von der Kommandozeile einlesen könnten. Wie das geht, wird beim Izhikevich-Modell für Simulations-Zeit und Schrittweite gezeigt.

Wenn Sie das Programm fertiggestellt haben, erzeugen Sie Ausgabe-Daten über die Ausgabe-Methode, und erstellen einen Plot der Aktivität des Neurons, zum Beispiel mit Octave oder Gnuplot. Versuchen Sie, die Parameter sinnvoll einzustellen, so daß phänomenologisch korrekte Ergebnisse produziert werden. Schicken Sie Ihre beiden Quelldateien per Email an den Tutor, und geben Sie sie auch in gedruckter Form ab.

Sollte etwas nicht klappen oder Fehler beim kompilieren Auftreten, wenden Sie sich per Mail an den Übungsgruppenleiter oder an die Liste.

Die einzelnen Modelle (aufgeteilt nach Abgabe-Gruppen) sind Modelle, die Sie in der Vorlesung kennengelernt haben. Für eine detailliertere Beschreibung sehen Sie bitte in Ihrem Skript nach.

### Kalter/Smirnow

Das Integrate-and-Fire-Modell (IF) lautet nach Vorlesung:

$$\begin{aligned} \partial_t v(t) &= I + a - bv, \\ v(0) &= v_0, \\ v(t) &= c, \qquad \text{falls } v > v_{thresh}. \end{aligned}$$

Hierbei ist $I$ der Eingangstrom und $a, b, c$ und $v_{thresh}$ sind Parameter des Modells.

Das Modell kann mit physiologischen Parametern auch geschrieben werden als

$$\begin{aligned} \partial_t v(t) &= \tfrac{1}{\tau_m}\left(-v + v_r + R_m I\right), \\ v(0) &= v_0, \\ v(t) &= v_{reset}, \qquad \text{falls } v > v_{thresh}. \end{aligned}$$

Als Parameter hat M. van Rossum gesetzt: Widerstand $R_m = 1$, $v_{reset} = v_r = -70mV$ sind der Reset-Wert und das Ruhepotential des Neurons und $v_{thresh} = -50mV$ ist die Schwelle zur Spike-Generierung. $\tau_m = 20ms$ ist die Zeitkonstante des Neurons, ausserdem kann man $v_0 = v_r$ als Anfangsbedingung setzen. $I$ ist der applizierte Strom.

Verwenden Sie das physiologische Modell.

### Lörwald

Das Integrate-and-Fire-Modell mit Adaption erweitert das obige IF-Modell und lautet:

$$\begin{aligned} \partial_t v(t) &= I + a - bv + g(d - V), \\ \partial_t g(t) &= \frac{e\delta(t) - g(t)}{\tau}, \\ v(0) &= v_0, \quad g(0) = g_0. \end{aligned}$$

Hierbei ist $I$ der Eingangstrom und $a, b, c, d, e$ sind Parameter. Das Modell erweitert das Integrate-and-Fire-Modell um einen $K^+$-Kanal, der die Aktivierungsdynamik im Modell beschreibt. $\tau$ ist die Zeitkonstante des $K^+$-Kanals, $\delta(t)$ eine Delta-Distribution. Ein Spike generiert einen Ausweichstrom, der die AP-Frequenz abschwächt.

### Hauck

Das Integrate-and-Fire-or-Burst (IFB)-Modell lautet:

$$\begin{aligned} \partial_t v(t) &= I + a - bv + g \cdot H \cdot (v - v_h) \cdot h(v_T - v), \\ \partial_t h(t) &= \begin{cases} \frac{-h(t)}{\tau^-} & \text{falls } v \geq v_h, \\ \frac{1-h(t)}{\tau^+} & \text{falls } v > v_h, \end{cases} \\ v(0) &= v_0, \quad h(0) = h_0, \\ v(t) &= c, \quad \text{falls } v > v_{thresh}. \end{aligned}$$

Das IFB-Modell erweitert IF um Bursting. Hierbei ist $I$ der Eingangstrom und $a, b, c, g$ und $v_{thresh}$ sind Parameter. $H$ ist eine im Paper von Smith und Cox, 2000 (*Fourier Analysis of Sinusoidally Driven Thalamocortical Relay Neurons and a Minimal Integrate-and-Fire-or-Burst-Model*) beschriebene Funktion. $h$ beschreibt einen Calcium-Strom, der Bursting ermöglicht. $\tau$ ist eine Zeitkonstante.

Im oben genannten Paper ist die physiologische, nicht reskalierte Version des Modells zu sehen:

$$\begin{aligned} C\partial_t v &= I + g_L(v - v_L) - g_T H \cdot (v - v_h) \cdot h(v_T - v), \\ \partial_t h(t) &= \begin{cases} \frac{-h(t)}{\tau^-} & \text{falls } v \geq v_h, \\ \frac{1-h(t)}{\tau^+} & \text{falls } v > v_h, \end{cases} \\ v(0) &= v_0, \quad h(0) = h_0, \\ v(t) &= c, \quad \text{falls } v > v_{thresh}. \end{aligned}$$

Implementieren Sie die physiologische Variante des Modells und orientieren Sie sich an der Parameterwerten aus oben genanntem Paper. Dort finden Sie auch eine Erklärung aller Funktionen und Parameter.

### Klein/Müller

Das Resonate-and-Fire-Modell (RF) lautet:

$$\begin{aligned} \partial_t z(t) &= I + (b + i\omega)z, \\ z(0) &= z_{init}, \\ z(t) &= z_0(z), \quad \text{falls } Im(z) = a_{th}. \end{aligned}$$

Hierbei ist der Realteil von $z$ das Membranpotential, $I$ der Eingangsstrom und $b, \omega$ und $a_{th}$ sind Parameter. Ist die Frequenz $\omega = 0$, reagiert das Modell wie ein Integrator. $z_0(z)$ ist eine Funktion zum aktivitätsabhängigen Reset nach einem Spike.

### Jeltsch/Pfeil

Das quadratische Integrate-and-Fire-Modell (QIF) lautet:

$$\begin{aligned} \partial_t v(t) &= I + a(v - v_{rest})(v - v_{thresh}), \\ v(0) &= v_0, \\ v(t) &= v_{reset}, \quad \text{falls } v = v_{peak}. \end{aligned}$$

Hierbei ist $I$ der Eingangstrom und $v_{rest}, v_{thresh}$ das Ruhepotential und der Schwellwert des Potentials. $v_{reset}$ ist der Wert, auf den das Potential nach Aktivität zurückgesetzt wird und $v_{peak}$ der Peak-Wert. $a$ ist ein Paramter.