Advanced Brian 2

(No fancy logo because this is serious.)

Installing

- Anaconda distribution
  - http://continuum.io/downloads
  - Works on Windows, Mac, Linux, 32/64 bit
  - No “make” utility on Windows
- Python(x, y) distribution
  - Windows 32 bit only (but can run on 64 bit machines)
  - Includes everything
- Both available on memory stick
- Brian 2 installation:
  - `pip install brian2 --pre` (fresh installation)
  - `pip install brian2 --pre --upgrade --no-deps` (upgrade)
- Will leave these instructions at the bottom of the screen

Runtime and standalone modes

- Runtime
  - Python / Numpy
  - C++ / Weave (doesn’t work on Python 3)
  - Cython (in progress)
- Standalone
  - C++
  - Android: Java / Renderscript (in progress)
  - GPU: GeNN (in progress)
  - GPU: NeMo (planned)
  - OpenCL (planned)
  - FPGA (planned)

Anaconda (http://continuum.io/downloads) or Python(x,y) [https://code.google.com/p/pythonxy/]

```
pip install brian2 --pre [--upgrade --no-deps]
```

Runtime code generation

- Select code generation target:
  - `brian_prefs.codegen.target = 'numpy'`
  - `brian_prefs.codegen.target = 'weave'`

- Demo

Custom functions

- Standard functions built in (sin, cos, etc.)
- To use your own function in Python/numpy mode:
  - Just declare the units with a decorator:

  -
    ```
    @check_units(t=second, result=1)
    def f(t):
        return t/second
    ```
- In C++/weave mode you have to add ‘support code’:

  -
    ```
    @make_function(codes={‘weave’:{‘support_code’:
    your_code_here}})
    ```

- Demo

Standalone code generation

- Select target device at beginning of script:
  - `set_device('cpp_standalone')`

  -
    ```
    import brian2genn
    set_device('genn')
    ```

- Build project at end of script

```
device.build(project_dir='STDP_standalone',
             compile_project=True,
             run_project=True)
```

- Demo

Limitations of standalone and workarounds

- Python code is not translated
  - `neurons.v = rand(N)*(Vt-Vr)+Vr`
  - Each time the compiled project is run the values will be the same
  - All that Brian sees is `neurons.v = an_array_of_values`
- Solution: use string-based initialization
  - `neurons.v = 'rand()*(Vt-Vr)+Vr'`

Limitations of standalone and workarounds

- Insert custom code directly into generated code
  - `device.insert_device_code(‘main’, code)`
  - Demo

- Interface with your own code
  - Insert Brian code into a function other than main
  - `with device.run_function(‘your_function_name’):`
  - Write your own main function
  - Demo

Extending Brian 2: new languages/devices

- Won't go through all the details (depending on time)
- Write a new language generator
  - Syntax translation (using `NodeRenderer`)
  - Translate basic language elements using `CodeGenerator`
    - Data types, scalars, constants, arrays, dynamic arrays
  - Implement templates for supported Brian objects
- Write a new runtime mode
  - Write a language generator if necessary
  - Implement the `CodeObject` (handles compiling, running, etc.)
- Write a new device
  - Write a language generator if necessary
  - Implement a `Device` object
- May not be as much work as it seems! (e.g. C++ already done)