

Contents:

1 API Reference	1
1.1 Package root	1
1.2 Calculation	1
1.3 Configuration	2
1.4 Simulation	2
1.5 Problems package	3
1.6 Poisson Problem	3
1.7 Utilities	3
Python Module Index	5

Chapter 1

API Reference

1.1 Package root

1.2 Calculation

`neurofem.calc.calc_mse(matrix, rhs, solution)`

Calculate the mean squared error (MSE) between Ax and b .

Parameters

- **matrix** (ndarray) – The system matrix A .
- **rhs** (ndarray) – The right-hand side vector b .
- **solution** (ndarray) – The estimated solution vector x .

Return type

float

Returns

The MSE value.

`neurofem.calc.calc_relative_residual(matrix, rhs, solution)`

Calculate the relative residual $\|Ax - b\| / \|b\|$.

Parameters

- **matrix** (ndarray) – The system matrix A .
- **rhs** (ndarray) – The right-hand side vector b .
- **solution** (ndarray) – The estimated solution vector x .

Return type

float

Returns

The relative residual value.

`neurofem.calc.calc_rmse(matrix, rhs, solution)`

Calculate the root mean squared error (RMSE) between Ax and b .

Parameters

- **matrix** (ndarray) – The system matrix A .

- **rhs** (ndarray) – The right-hand side vector b.
- **solution** (ndarray) – The estimated solution vector x.

Return type
float

Returns
The RMSE value.

1.3 Configuration

```
class neurofem.config.NeurofemConfig(gamma=50, dt=0.000244140625, theta=None, lambda_d=0.2,  
                                      lambda_v=0.4, k_p=4.0, k_i=16.0, sigma=0.00225,  
                                      steady_state=0.4)
```

Bases: object

A class to hold configuration parameters for NeuroFEM simulations. This only includes parameters specific to the NeuroFEM algorithm.

```
class neurofem.config.SpiNNakerConfig(s2_ip=None, stm_ip=None, is_multi_node_board=False)
```

Bases: object

This class holds configuration parameters for SpiNNaker2 hardware. It supports both single-chip and multi-node (48-chip) boards.

```
CHIPS_PER_BOARD: int = 48
```

```
CORES_PER_CHIP: int = 148
```

```
get_hardware()
```

Get the py-spinnaker2 hardware object based on the configuration.+ Note that this requires the spinnaker2 package to be installed.

Return type

hardware.SpiNNaker2Chip | hardware.SpiNNcloud48NodeBoard

Returns

SpiNNaker2 hardware object

```
property is_multi_node_board: bool
```

Whether using a multi-node (48-chip) board.

```
property max_cores: int
```

Maximum number of cores available on the board.

```
property s2_ip: str | None
```

The IP address of the SpiNNaker2 chip (for single-chip boards).

```
property stm_ip: str | None
```

The IP address of the STM controller (for multi-node boards).

1.4 Simulation

```
class neurofem.simulation.NeurofemSimulation(spinn_config, config, matrix, rhs)
```

Bases: object

Class representing a NeuroFEM run on SpiNNaker hardware.

```
run(timesteps=50000.0, sys_tick_in_s=0.001, mapping_only=False)
```

Run the NeuroFEM simulation on SpiNNaker hardware and retrieve the solution.

Parameters

- **timesteps** (int) – Number of simulation timesteps to run.
- **sys_tick_in_s** (float) – System tick duration in seconds.
- **mapping_only** – If True, only perform the mapping without running.

Return type

ndarray

Returns

The solution vector as a NumPy array.

1.5 Problems package

1.6 Poisson Problem

```
neurofem.problems.poisson.generate_poisson_unitdisk_variable_f(nrefs=5)
```

Assemble the linear system Ax=b for a Poisson problem on the unit disk. The continuous problem is:

-u = f in unit disk, u=0 on boundary

with:

$$f(x,y) = 12 - 60 * (x - 0.25)^2 - 60 * (y + 0.13)^2$$

Parameters

nrefs (int) – Number of mesh refinements.

Returns

A tuple (matrix, rhs, basis) where - matrix is the assembled stiffness matrix A (CSR format) - rhs is the assembled right-hand side vector b - basis is the finite element basis object

1.7 Utilities

```
neurofem.problems.utils.float_to_signed_sparse(matrix, x_bits=21, scale=None)
```

Quantize a sparse matrix to signed integers with specific bit-width.

The input floating point matrix is converted to a sparse integer matrix using uniform quantization: quantized_value = round(original_value / scale)

The range of representable integers is determined by the number of bits specified (x_bits). One bit is reserved for the sign. For example, for x_bits=6, the representable range is from -32 to 31.

Parameters

- **matrix** (spmatrix | ndarray) – The input sparse matrix.
- **x_bits** (int) – Number of bits for the quantized integers (including sign bit).
- **scale** (float | None) – Optional scale factor for quantization. The matrix is divided by this value before rounding. Will be computed if not provided.

Return type

tuple[spmatrix, float]

Returns

The quantized CSR matrix of type int32 and the scale factor.