MMODEL

modular modeling framework for scientific prototyping

modular
- DAG workflow
- based on networkx

lightweight
- function as node

distributable
- with graph or model

customizable
- wrapper as modifier

addresses:
- different programming proficiencies
- rapid prototyping
- difficulties in unit tests

Peter Sun, John Marohn
Department of Chemistry and Chemical biology, Cornell University

Overview

```python
def func_a(x, y):
    return x + y


def func_b(sum_xy, z):
    return math.log(sum_xy, z)


def func_c(sum_xy, log_xy):
    return sum_xy * log_xy


doc = """MModel example function
calculates the mathematical formula (x + y)log(x + y, z)
"""
graph = ModelGraph(name="Example", doc=doc)

grouped_edges = [("func a", ["func b", "func c"]), ("func b", "func c")]
graph.add_grouped_edges_from(grouped_edges)

node_objects = [
    ("func a", func_a, ["sum_xy"]),
    ("func b", func_b, ["log_xy"]),
    ("func c", func_c, ["result"]),
]
graph.set_node_objects_from(node_objects)

example_func = Model(graph, handler=MemHandler)
```

define functions

define graph

define nodes and edges

link nodes to functions

define model

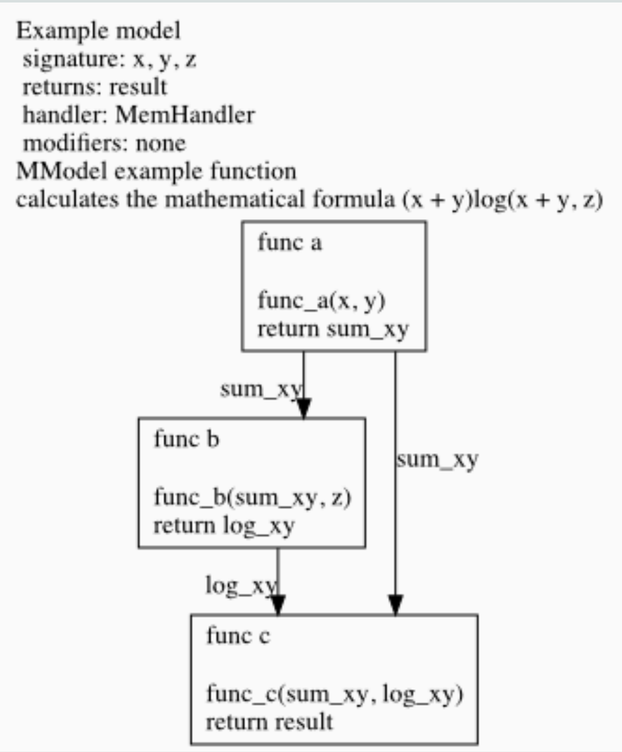
```
>>> print(example_func)
Example model
  signature: x, y, z
  returns: result
  handler: MemHandler
  modifiers: none
MModel example function
Calculates the mathematical formula (x + y)log(x + y, z)

>>> example_func(5, 3, 2) # (5 + 3)log(5 + 3, 2)
24.0

>>> example_func.draw() # draw model graph
```

MModel components

graph: outlines DAG workflow (networkx based)
handler: handles graph execution
modifier: decorates graphs, subgraphs, or nodes
model: creates workflow callables



MMODEL

modular modeling framework for scientific prototyping

modular
- DAG workflow
- based on networkx

lightweight
- function as node

distributable
- with graph or model

customizable
- wrapper as modifier

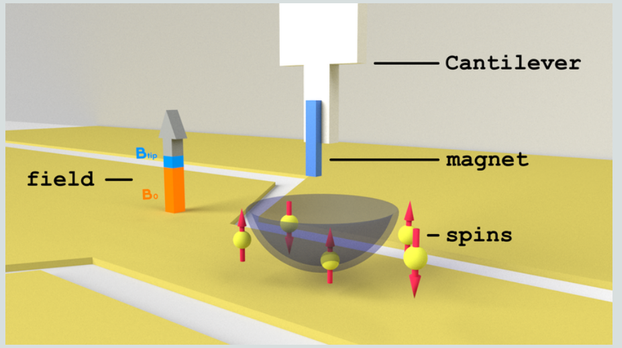
addresses:
- different programming proficiencies
- rapid prototyping
- difficulties in unit tests

Peter Sun, John Marohn
Department of Chemistry and Chemical biology, Cornell University

Building scientific packages using mmodel (developer)

mrfmsim

Simulate signals in Magnetic Resonance Force Microscope experiments



Create a new experiment:

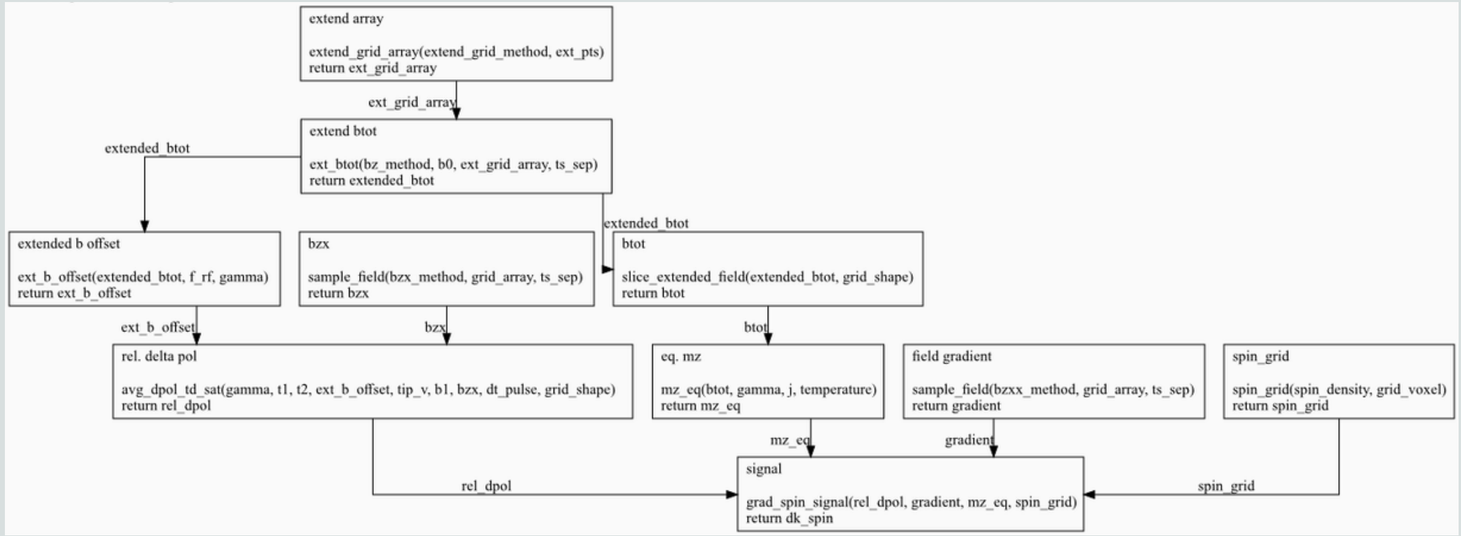
```python
from mmodel import ModelGraph
from mrfmsim.function import ...

grouped_edge_list = [
    ("extend array", "extend btot"),
    ("extend btot", ["extended b offset", "btot"]),
    ("btot", "eq. mz"),
    (["bzx", "extended b offset"], "rel. delta pol"),
    (["eq. mz", "field gradient", "rel. delta pol", "spin_grid"], "signal"),
]

td_esr_graph = ModelGraph(doc=doc, name = "td_esr")
td_esr_graph.add_grouped_edges_from(grouped_edge_list)
```

```python
node_objects = [
    ("extend array", extend_grid_array, ["ext_grid_array"]),
    ("extend btot", ext_btot, ["extended_btot"]),
    ("btot", slice_extended_field, ["btot"]),
    ("bzx", sample_field, ["bzx"], [smod(["bzx_method", "grid_array", "ts_sep"])]),
    ("field gradient", sample_field, ["gradient"]),
    ("eq. mz", mz_eq, ["mz_eq"]),
    ("rel. delta pol", avg_dpol_td_sat, ["rel_dpol"]),
    ("extended b offset", ext_b_offset, ["ext_b_offset"]),
    ("spin_grid", spin_grid, ["spin_grid"]),
    ("signal", grad_spin_signal, ["dk_spin"]),
]

td_esr_graph.set_node_objects_from(node_objects)
ts_esr_model = Model(td_esr_model, handler=MemHandler, modifiers=[])
```



Developer role

1. define functions
2. **define experiment graphs with functions**
3. **define default models**
4. test functions and models
5. define shortcuts and additional modifiers

MMODEL

modular modeling framework for scientific prototyping

modular
- DAG workflow
- based on networkx

lightweight
- function as node

distributable
- with graph or model

customizable
- wrapper as modifier

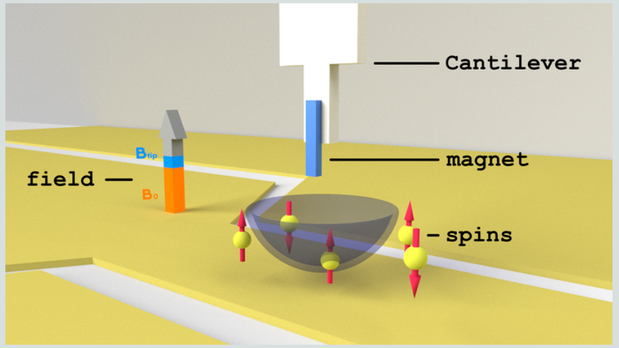
addresses:
- different programming proficiencies
- rapid prototyping
- difficulties in unit tests

Peter Sun, John Marohn
Department of Chemistry and Chemical biology, Cornell University

Building scientific packages using mmodel (developer)

mrfmsim

Simulate signals in Magnetic Resonance Force Microscope experiments



Create a package-specific shortcut:

mmodel
```
subgraph = subgraph_by_parameters(graph, ["b0"])
loop_node = Model(subgraph, MemHandler, [loop_modifier("b0")])
looped_graph = modify_subgraph(graph, subgraph, "b0 loop node", loop_node)
looped_model = Model(looped_graph, handler=MemHandler)
```

mrfmsim
```
looped_model = loop_shortcut(td_esr_model, "b0")
```

mrfmsim
```
from mmodel import Model, subgraph_by_parameters, modify_subgraph, loop_modifier

def loop_shortcut(model, loop_parameter: str):
    """Shortcut to add loop to subgraph of the given model

    :return: a new model that loops the parameter in the subgraph
    """
    loop_mod = loop_modifier(loop_parameter)
    handler = model._handler
    graph = model._graph
    model_modifiers = model._modifiers
    node_name = f"{loop_parameter} loop node"

    subgraph = subgraph_by_parameters(graph, [loop_parameter])
    loopee_node = Model(subgraph, handler, loop_mod)
    looped_graph = modify_subgraph(graph, subgraph, node_name, loop_node)
    looped_model = Model(looped_graph, handler)

    return looped_model
```

Modify a subgraph in mmodel

1. find the subgraph
2. create a "function" (model) base on subgraph
3. substitute graph with a "submodel"
4. create a new model based on the new graph

Developer role

1. define functions
2. define experiment graphs with functions
3. define default models
4. test functions and models
5. **define shortcuts and additional modifiers**

MMODEL

modular modeling framework for scientific prototyping

modular
- DAG workflow
- based on networkx

lightweight
- function as node

distributable
- with graph or model

customizable
- wrapper as modifier

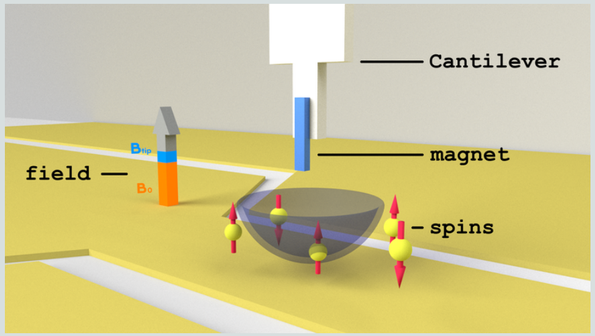
addresses:
- different programming proficiencies
- rapid prototyping
- difficulties in unit tests

Peter Sun, John Marohn
Department of Chemistry and Chemical biology, Cornell University

Building scientific packages using mmodel (user)

mrfmsim

Simulate signals in Magnetic Resonance Force Microscope experiments



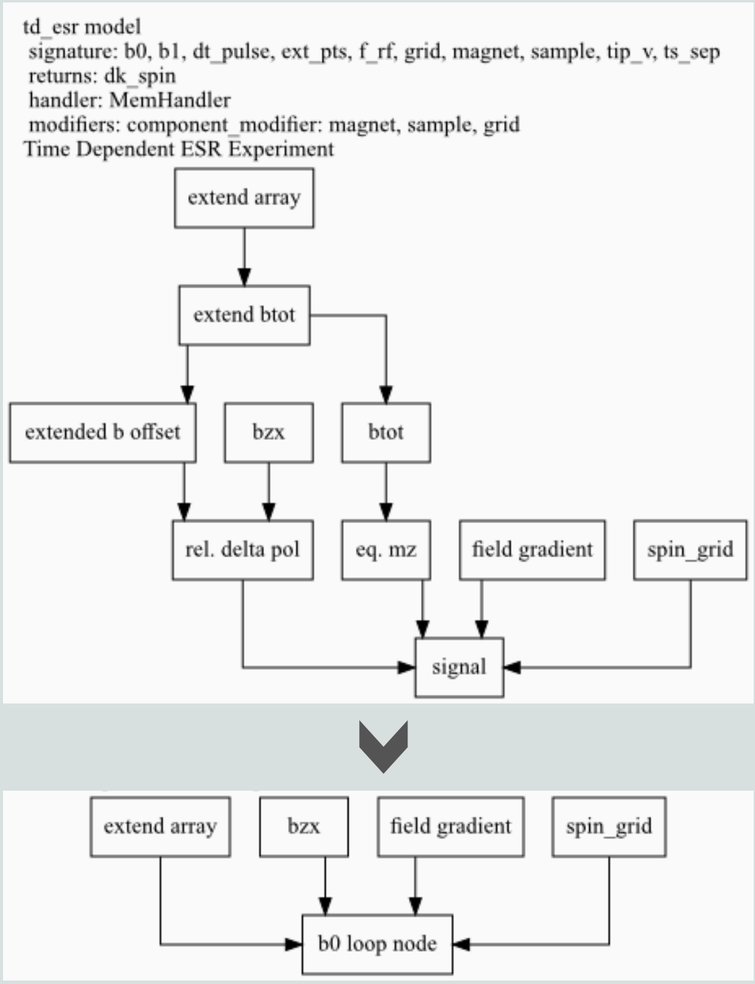
Create a loop in model and execute the model:

```python
from mmodel import draw_plain_graph
from mrfmsim.shortcut import loop_shortcut
from mrfmsim.experiment import td_esr_model

td_esr_model.draw(method=draw_plain_graph)

td_esr_b1_looped_model = loop_shortcut(td_esr_model, "b1")

dk_spin_td_b1loop = td_esr_b1_looped_model(
    dt_pulse=5.44e-4,
    grid=grid,
    magnet=magnet,
    sample=sample,
    ts_sep=[0, 150.0, 0],
    b1=b1_list,
    f_rf=18.1e9,
    b0=645,
    ext_pts=[x_pts, 0, 0],
    tip_v=tip_v,
)
```



User role

1. Inspect model graph
2. Modify model or subgraph using modifiers
3. Execute model as a function