Technical specification

Solvers

Ordinary differential equations (ODE)

Morpheus implements explicit finite different solvers of numerical integration of ODEs:

- Euler's method: 1st order
- Heun's method (a.k.a. midpoint): 2nd order
- Runge-Kutta's method: 4th order

and stochastic differential equations (SDEs):

- Euler-Maruyama's method for 1st and 2nd order.

The latter is automatically used when a stochastic white noise term `rand_norm(0,[amplitude])` is included as below, but is not available for Runge-Kutta.

```
<DiffEqn solver="heun" time-step="0.01" symbol-ref="X">
  <Expression> rand_norm(0, noise) </Expression>
</DiffEqn>
<Constant symbol="noise" value="1e-3"/>
```

Fixed time stepping

Solvers with adaptive time stepping, such as the Runge-Kutta-Fehlberg method, are not available. Currently, we use fixed time stepping methods, because this greatly simplifies temporal scheduling of updates for automated model integration.

Stiff systems

Although the implemented explicit solvers are general and flexible, they are not applicable to stiff systems. Stiff systems are ODE systems that can change so rapidly that explicit solvers will result in numerical instability, unless the time step is extremely small. This situation may occur in systems where very large and small values are multiplied.

Lattice

Lattice structure

Morpheus is a lattice-based simulation platform. This means that the spatial models are discretized on

a lattice.

The following lattices are available:

- **linear** 1D lattice
- **square** 2D (orthogonal) lattice
- **hexagonal** 2D lattice
- **cubic** 3D (orthogonal) lattice

Boundary conditions

See description in FAQ

Membrane Properties

For `MembraneProperties`, properties that are resolved on the surface of cells, Morpheus uses a special lattices with polar coordinate system

- **circular** 1D lattice (for 2D models)
- **spherical** 2D lattice (for 3D models)

Diffusion

Method of lines

Partial differential equations (PDEs) are numerically approximated by separating the reaction and diffusion steps using the method of lines. This method discretizes the PDE into a system of coupled ODEs. These ODEs are then solved using the numerical methods mentioned above.

Diffusion equation

For diffusion, Morpheus uses the simple and general forward Euler scheme. The time step δt is automatically adjusted according to the CFL condition to guarantuee numerical stability. Alignment of multidimensional lattices to 1D memory (using valarrays) results in highly efficient implementation of diffusion.

The unconditionally stable method of alternate direction implicit (ADI) is also implemented (for constant (Dirichlet) boundary conditions), but is currently not used.

Motility

Cellular Potts model

Cell shape and motility is implemented according to the cellular Potts model (CPM)(Graner and Glazier, 1992).

In this model formalism, biological cells are represented as a domains of lattices sites $\boldsymbol{x}$ with identical index σ. In its simplest form (ignoring differential adhesion), the changes in configuration of cells on the lattice are governed by the Hamiltonian:

$H=\sum_{\sigma > 0} \lambda_V (v_\sigma - V_t)^2 + \sum_{\sigma > 0} \lambda_P (p_\sigma - P_t)$

where v_σ and p_σ are the actual volume and perimeter of the cell with index σ and V_t and P_t are the target volume and perimeter. Deviations from these target values increase the *free energy* H according to the scalars λ_V and λ_P.

Monte Carlo

The CPM is a Monte Carlo method in which the lattice is updated by randomly sampling lattice sites.

Within the CPM, a Monte Carlo step (MCS) is often taken as a discrete unit of time. A single Monte Carlo step is defined as the number of random sampled updates equal to the number of lattice sites, i.e. within one step, each lattice sites would have had chance to be updated.

Metropolis kinetics

In each update:

1. A lattice site $\boldsymbol{x}$ is chosen at random.

2. From the neighborhood N of $\boldsymbol{x}$, a second lattice site $\boldsymbol{x'}$ is chosen.

3. Then, the change in *free energy* ΔH is calculated *if* the state σ at $\boldsymbol{x'}$ ($\sigma_\boldsymbol{x'}$) would be copied to $\boldsymbol{x}$ according to the specific Hamiltonian H.

4. The proposed update is always accepted when $\Delta H<0$ and is accepted with a Boltzmann probability when $\Delta H>0$:

$P(\sigma_\boldsymbol{x'} \rightarrow \sigma_\boldsymbol{x})=\begin{cases} 1 &\mbox{if } \Delta H<0 \\ e^{{(\Delta H-Y)}/T} &\mbox{otherwise} \end{cases}$

where T (for 'temperature') modulates the probability of unfavorable updates to be accepted and can be taken to represent local protrusions/retractions of the cell membrane. The parameter Y (for 'yield') is sometimes used to avoid oscillations with energy-neutral updates. This can be said to represent cytoskeletal resistance to membrane fluctuations.

Random numbers

By default, Morpheus uses the Mersenne Twister 19937 pseudo-random number generator included in the CPP GNU compiler (TR1). Alternatively, the Boost RNG can be used (specified in CMake options).

Note: In multithreaded simulations, each thread gets its own RNG (the random seeds for which are based on the specified seed of the master thread RNG). Therefore, to reproduce simulation results, not only the random seed needs to be specified (`Time/RandomSeed`), but also the same number of threads (`Settings → Local→ threads`).

Plugins

Morpheus provides a plugin interface to extend the feature set. Plugins are written in C++ and requires recompilation from source. Therefore, this type of extensibility is only available when building from source (currently limited to developers and collaborators).

Morpheus has interfaces for different types of plugins:

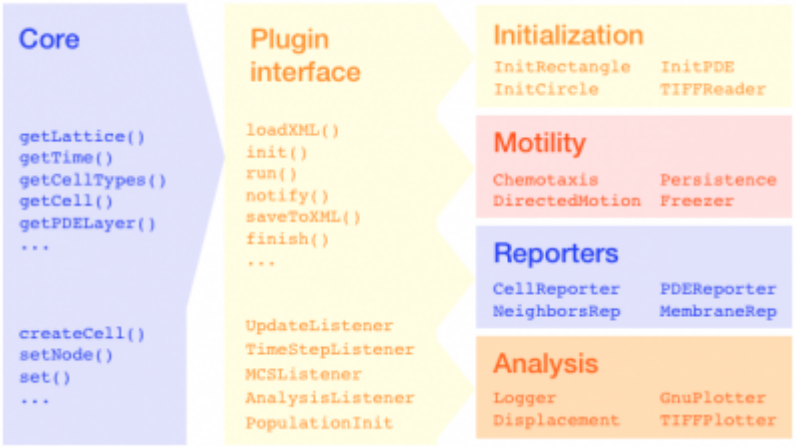
- **Initialization**: Plugins with methods to define an initial condition. Examples: `TIFFReader` and `InitPDEExpression`.
- **Reporters**: Plugins with customizable spatial mappings used in situations in which spatial mapping cannot be automated due to ambiguity. Examples are `NeighborsReporter` and `PDEReporter`.
- **Motility**: Plugins with methods for cell movement. Examples: `Chemotaxis` and `Persistence`.
- **Shape**: Plugins with methods to constrain cell shape. Examples: `VolumeConstraint` and `SurfaceConstraint`.
- **Analysis**: Plugins for data logging, plotting or processing. Examples: `Gnuplotter` and `HistogramLogger`.
- **Miscellaneous**: Set of plugins that do not fit in the categories above. Examples: `Proliferation` and `InsertMedium`.

Writing new plugins requires you to add three files:

- Header (MyPlugin.h): declares variables and functions
- Implementation (MyPlugin.cpp): implements the core functions
- XSD (MyPlugin.xsd): specifies the rules what constitutes valid XML input (used by GUI)

See an example for an `Analysis` plugin below.

Plugin interfaces



Plugin interface provides interfaces for various types of plugins.

Header file (MyPlugin.h)

```
<Analysis>
   <MyPlugin celltype="cells" symbol="s" interval="1.0"/>
<Analysis>
```

```
#include "core/interfaces.h"
#include "core/simulation.h"
#include "core/celltype.h"

class MyPlugin : public Analysis_Listener
{

public:
    DECLARE_PLUGIN("MyPlugin");
    NetworkLogger(){};
    virtual void loadFromXML(const XMLNode);
    virtual void notify(double time);
    virtual void init(double time);

private:
    string symbolstr, celltypestr;
    SymbolAccessor<double> symbol;
    shared_ptr<const CellType> celltype;
};
```

Inherit interface

Override functions

Symbol references

example header file

Implementation (MyPlugin.cpp)

```
  #include "myPlugin.h"
REGISTER_PLUGIN(MyPlugin);

void MyPlugin::loadFromXML(const XMLNode Node)
{
   Analysis_Listener::loadFromXML( Node );
   getXMLAttribute(Node,"celltype", celltypestr);
   getXMLAttribute(Node,"symbol", symbolstr);
}

void MyPlugin::init(double time)
{
    Analysis_Listener::init(time);
    celltype = CPM::findCellType(celltypestr);
    symbol   = SIM::findSymbol<double>(symbolstr, celltype);
}

void MyPlugin::notify(double time)
{
    Analysis_Listener::notify(time);
    for(uint c=0; c < celltype->getCellIDs().size(); c++){
       double value = CPM::getCell( cells[c] ).get( symbol );
       // do something
    }
    // write to file
}
```

Read parameters

Initialize references

Use symbols

example implementation file

XML Schema (MyPlugin.xsd)

```
<Analysis>
  <MyPlugin interval="1.0" celltype="cells" symbol="s"/>
<Analysis>
```

```
<xs:schema>
   <xs:complexType name="MyPlugin">
      <xs:annotation>
         <xs:documentation>Text shown in GUI</xs:documentation>
      </xs:annotation>
      <xs:attribute name="interval" type="cpmDouble"          use="required"/>
      <xs:attribute name="celltype" type="cpmCellTypeRef"     use="optional"/>
      <xs:attribute name="symbol"   type="cpmDoubleSymbolRef" use="required"/>
   </xs:complexType>
</xs:schema>
```

Documentation

Attributes

example XML schema file