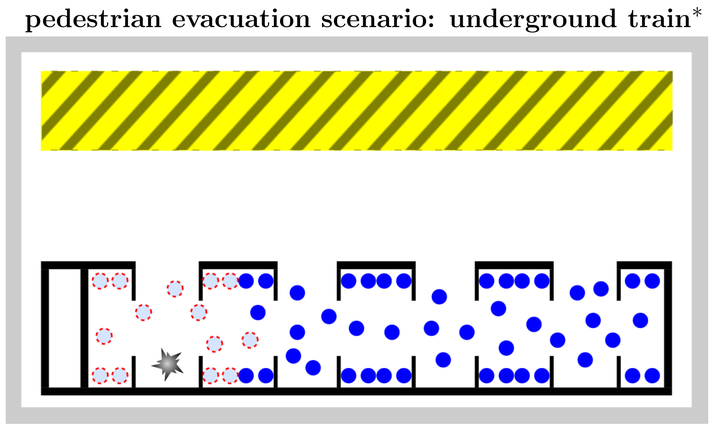
Practical use of Chaospy for pedestrian traffic simulations

Florian Künzner[§], Tobias Neckel[§], Isabella von Sivers[§,‡]
[§]Technical University of Munich, [‡]Munich University of Applied Sciences

pedestrian traffic simulation:

- simulation of an evacuation scenario:
  - inspired by the London bombings 2005
  - limited to 1 car underground train with 60 passengers
- VADERE (Java), pedestrian traffic, movement, and evacuation simulator:
  - optimal steps model (OSM) [3]: pedestrian movement
  - social identity model algorithm (SIMA) [2]: helping behaviour
- goal: understand the influence of the uncertain parameters on the scenario
- UQ method: stochastic collocation with the pseudospectral approach
- quantity of interest (QoI):
  - evacuation time
  - pedestrians evacuation over time steps

pedestrian evacuation scenario: underground train*



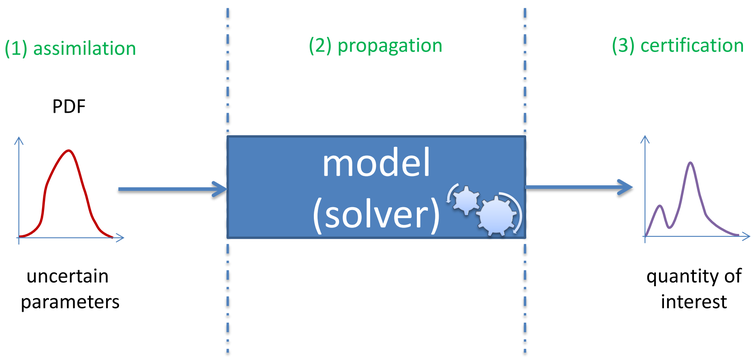
summary

Chaospy:
- is suitable for prototyping: fast development
- is working with black-box solvers for forward UQ
- is flexible, extensible and has the source on github
- has good support through the developers

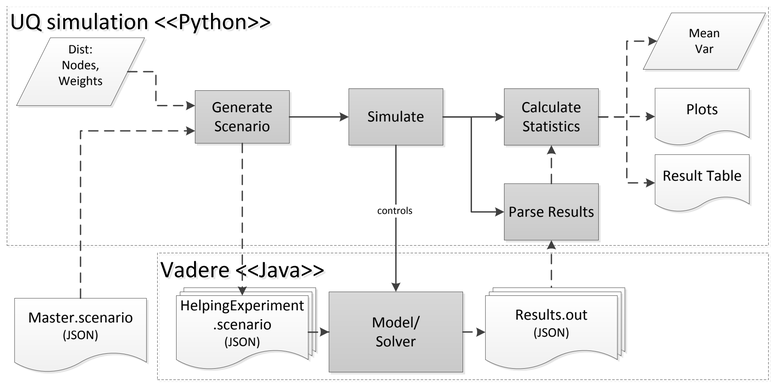
Python:
- offers easy pre-/postprocessing of data
- has good support for parallel execution on clusters

pedestrian simulation:
- VADERE simulator: works well as black-box solver
- application scientists can now better understand the behaviour of their algorithms

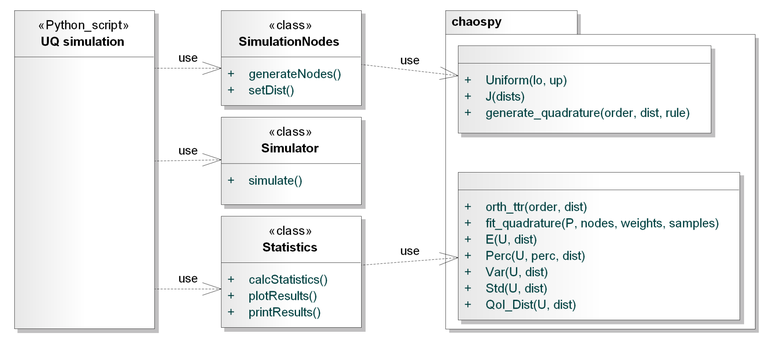
uncertainty quantification phases*



uncertainty quantification simulation pipeline



simulation program architecture



(1) assimilation

determine the uncertain parameters

parameters of social identity model algorithm*

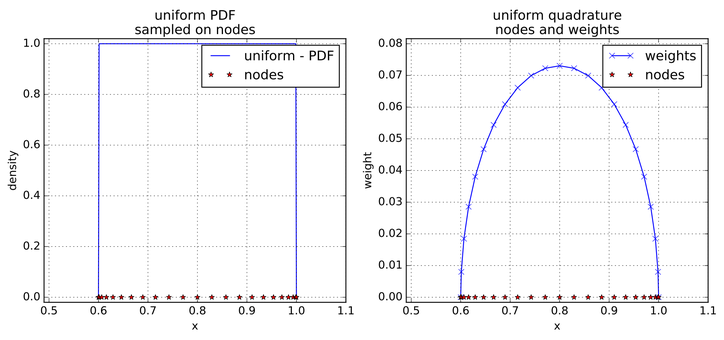
parameter	description	uniform dist. min.	uniform dist. max.
$p_{sharingSI}$	people sharing a social identity	0.6	1.0
$p_{injPeds}$	percentage of injured pedestrians	0.1	0.3
v_{inj}	speed of a helper with an injured pedestrian (m/s)	0.4	0.8

```
#generate distributions
p_sharingSI_dist = cp.Uniform(0.6, 1.0)
p_injPeds_dist = cp.Uniform(0.1, 0.3)
v_inj_dist = cp.Uniform(0.4, 0.8)

#join the dists
parameter_dist = cp.J(p_sharingSI_dist, p_injPeds_dist, v_inj_dist)
```

generate stochastic collocation nodes and weights

```
#generate collocation nodes and weights
#-> #21 nodes per parameter; -> #9126 combinations for all 3 parameters
nodes, weights = cp.generate_quadrature(q_order, parameter_dist,
                                        rule="Gaussian")
```

parameter: people sharing a social identity

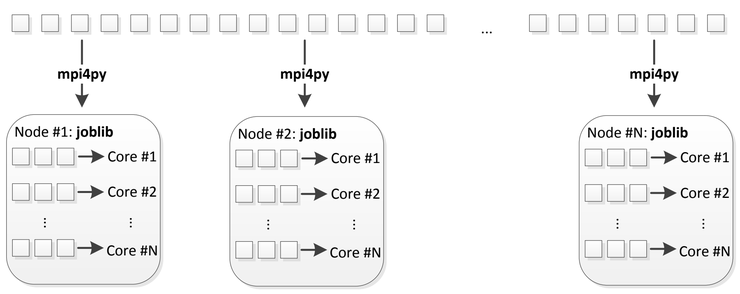


(2) propagation

call the black-box solver

```
def solver(perc_sharingSI, perc_injPeds, v_inj):
    #create HelpingExperiment.scenario based on collocation nodes
    config = json.load(open("Master.scenario"))
    #... manipulate scenario file with parameters
    json.dump(config, open("HelpingExperiment.scenario", 'w'))

    #execute simulator
    systemCommand = "java -jar vadare.jar HelpingExperiment.scenario"
    ret = os.system(systemCommand)

    #read quantity of interest
    evacuationTime = readEvacuationTime("Results.out")

    return evacuationTime
```

parallelise the propagation




```
#distribute collocation nodes with MPI
if rank == 0:
    nodesToSend = np.array_split(nodes, num_nodes)

MPI.COMM_WORLD.scatter(nodesToSend, nodes)

#parallelisation with joblib on one computation node on the cluster
samples = Parallel(n_jobs=num_cores)(
              delayed(solver)(node[0], node[1], node[2])
              for node in nodes.T)

#collect results
MPI.COMM_WORLD.Gather(samples, all_samples)
```

(3) certification

how-to certification

- understand the impact of uncertainty in real world scenarios: all parameters uncertain
- understand the impact of a single parameter: only the particular parameter uncertain, others kept fixed
- understand the impact of a subset of the parameter uncertain: only the subset of parameters uncertain, others kept fixed

calculate statistics

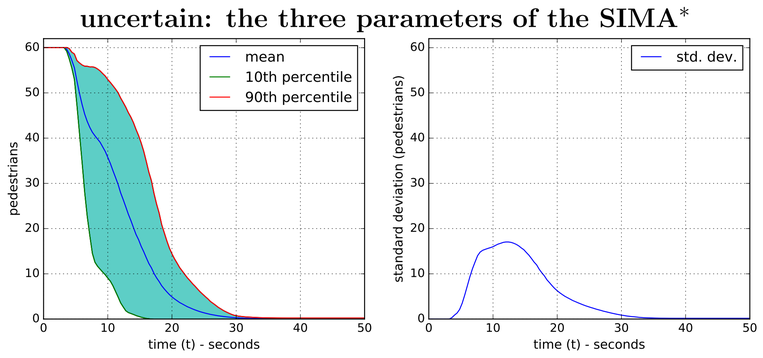
```
#genererate orthogonal polynomial
orth_poly = cp.orth_ttr(p_order, parameter_dist)

#calc polynomial chaos expansion
U = cp.fit_quadrature(orth_poly, nodes, weights, all_samples)

#expectation value (mean, E)
E = cp.E(U, parameter_dist)
#percentiles (Pxx)
P10 = cp.Perc(U, 10, parameter_dist, 10*5)
P90 = cp.Perc(U, 90, parameter_dist, 10*5)
#variance (Var)
Var = cp.Var(U, parameter_dist)
#standard deviation (std. dev.)
StdDev = cp.Std(U, parameter_dist)
#quantity of interst distributions (qoi_dists)
qoi_dist = cp.QoI_Dist(U, parameter_dist)
```

cp.QoI_Dist()

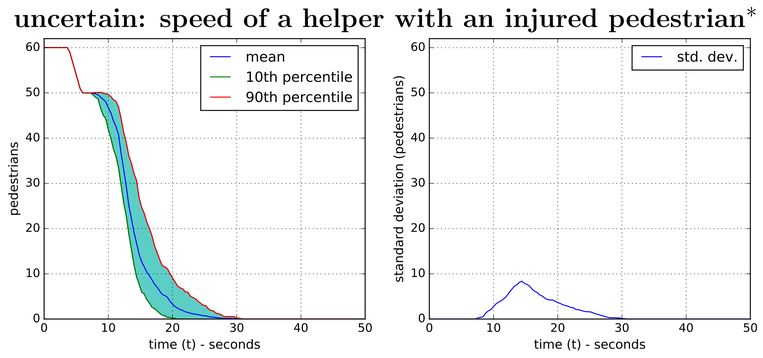
- constructs a distribution of the quantity of interest
- internal:
  - samples from `U`
  - uses a kernel density estimator (KDE)
- useful for:
  - creating 3D plots of the quantity of interest distribution
  - make a second UQ simulation with `qoi_dist` as input

visualisation of results

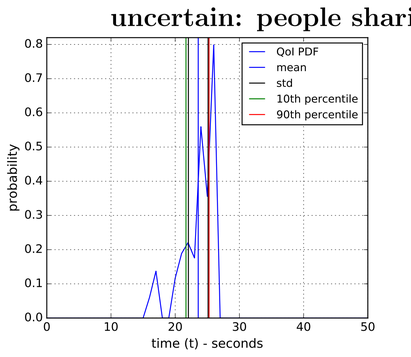
uncertain: the three parameters of the SIMA*

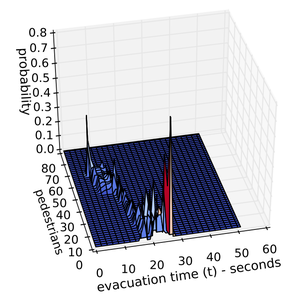


uncertain: speed of a helper with an injured pedestrian*

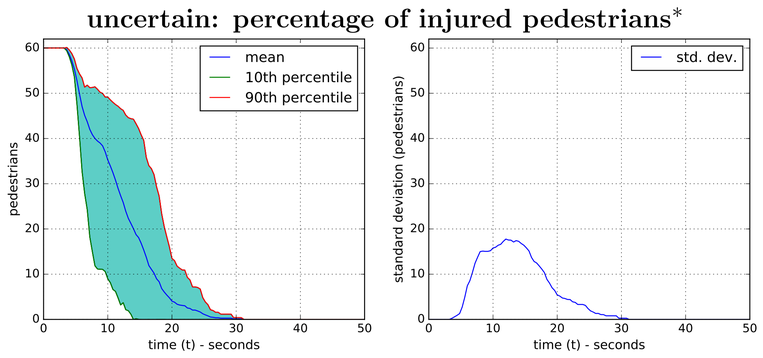


uncertain: people sharing a social identity

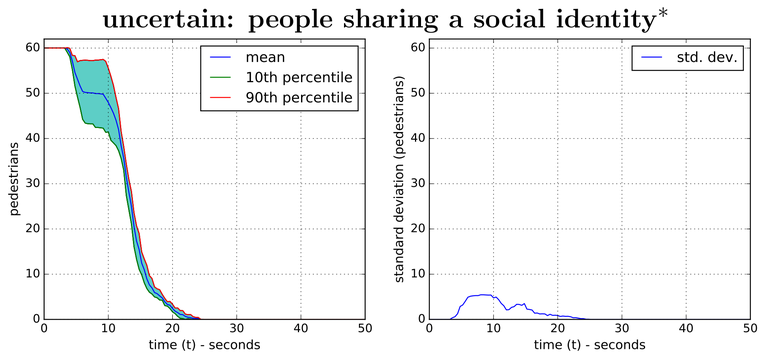




uncertain: percentage of injured pedestrians*



uncertain: people sharing a social identity*



comparison of mean evacuation time*

uncertain parameter	evacuation time mean	evacuation time std. dev.
all three parameters	21.35	5.68
people sharing a social identity	21.62	1.11
percentage of injured pedestrians	**20.60**	**5.43**
speed of a helper with an injured pedestrian	21.81	3.29

references

Chaospy:
- [1] Feinberg, J., Langtangen, H.P., 2015. Chaospy: An open source tool for designing methods of uncertainty quantification. Journal of Computational Science, Volume 11, Pages 46–57.
- source `https://github.com/hplgit/chaospy`

pedestrian simulation:
- [2] von Sivers, I., Templeton, A., Künzner, F., Köster, G., Drury, J., Philippides, A., Neckel, T., Bungartz, H.-J. Modelling social identification and helping in evacuation simulation. To appear in Safety Science.
- [3] Seitz M. J., Köster, G., 2012. Natural discretization of pedestrian movement in continuous space. Physical Review E 86, 046108.
- VADERE simulator (Munich University of Applied Science): coming soon open source

Python common libs:
- numpy `http://www.numpy.org`
- scipy `http://www.scipy.org`
- joblib `https://pypi.python.org/pypi/joblib`
- mpi4py `http://mpi4py.scipy.org`
- matplotlib `http://matplotlib.org`

* Figure taken from [2]