MCMC Example

Davide G.
in collaboration with UCL HR

Program

- Data vector
- Bayesian Statistic
- Synthetic Data + REAL world measurement
- MCMC implementation (emcee)

Data vector

How much stressed is a PhD. student as a function of time and weather?

Model Parameters:

1. Thesis anxiety
2. What am I doing?
3. Pub attendance
4. Finish the paper!
5. Sport activities

```python
"""
variables of the data vector function:
time from 0 to 36 months,
weather conditions
from - 0.3 (rain/cold/pollution --> Mordor ) to + 0.3 (sun --> Shire)
"""


"""""""""""""""""""""""""""""""""""""""""""""""""""
define data vector function,
theoretical model
"""""""""""""""""""""""""""""""""""""""""""""""""""
def phd_stress_function(time, weather, theta_ar, doing_sport):

    """ different factors  """
    thesis_anxiety   = theta_ar[0] * np.exp(time / 36.0)
    what_am_i_doing  = theta_ar[1] * np.exp(- time / 36.0)
    pub_attendance   = theta_ar[2] * ((time - 18.0) / 36.0) ** 2.0 + weather
    finish_the_paper = theta_ar[3] * (np.sin((5.0 * 2.0) * np.pi * time / 36.0) + 1.0)

    if doing_sport == True:
        sport_activities = theta_ar[4] * (- time / 36.0) + 1.0 + weather
        return thesis_anxiety + what_am_i_doing - pub_attendance \
                + finish_the_paper - sport_activities

    return thesis_anxiety + what_am_i_doing - pub_attendance + finish_the_paper
```

UCL

Unavoidable Bayesian Statistic Slide

- Prior: previous knowledge / degree of belief
- Likelihood: is the probability of the evidence given the parameters (wikipedia)
- Posterior: is the probability of the parameters given the evidence (again wikipedia)

Likelihood

Class Prior Probability

$$P(c \mid x) = \frac{P(x \mid c)\, P(c)}{P(x)}$$

Posterior Probability

Predictor Prior Probability

Pythonically speaking:

```python
""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""
define prior, likelihood, posterior functions
""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""
def lnprior(x,ranges):

    par_list = x

    for i in range(len(par_list)):
        if (par_list[i] < ranges[i,0]) or (par_list[i] > ranges[i,1]):
            return  -np.inf

    return 0.0

""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""
""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""
def lnlike(x, meas, params, inv_cov):

    par_list = x

    time_ar, weather, theta_ranges, doing_sport = params

    theta_ar = np.array(par_list)
    model    = phd_stress_function(time_ar, weather, theta_ar, doing_sport)
    diff     = model - meas

    return - np.dot(diff, np.dot(inv_cov, diff)) / 2.0
```

```python
""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""
""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""
def lnprob_f( x, meas, params, icov):

    time_ar, weather, theta_ranges, doing_sport = params

    lp = lnprior(x,theta_ranges)

    if not np.isfinite(lp):
        return - np.inf

    lnprob = lp + lnlike(x, meas, params, icov)

    # Check for lnprob returning NaN.
    if np.any(np.isnan(lnprob)):
        # Find indexes of lnprob array with NaN values.
        indxs_of_bad_p = np.where(np.isnan(lnprob) == True)[0]
        # Print some debugging stuff.
        print("NaN value of lnprob for parameters: ")
        print(x[indxs_of_bad_p])
        # Finally raise exception.
        raise ValueError("lnprob returned NaN.")

    return lnprob
```

N.B. uninformative prior

Main Code: synthetic data + REAL life PhD. student opinion

```python
""" spoiler alert: true values parameters """

true_theta_ar = np.array([1.0, 1.8, 2.7, 0.2, 0.6])
doing_sport   = False

""" when we asked the PhD. students their opinions and what was the weather like (coordinates) """
time_ar = np.linspace(1.0,36.0,72)
weather = np.cos(6.0 * np.pi * time_ar / 36.0  + np.pi/3.0 ) * 0.3

""" generate N independent / uncorrelated PhD. students opinions (SYNTHETIC DATA !!)"""
N_phd_students        = 200
true_stress           = phd_stress_function(time_ar, weather, true_theta_ar, doing_sport)
std_normal_dist       = np.ones(true_stress.size) * 0.1
N_survey_measurements = np.random.normal(true_stress, std_normal_dist,(N_phd_students,true_stress.size ))

""" covariance matrix derived from synthetic data """
cov_matrix            = np.cov(N_survey_measurements.T)

""" special PhD. student (MEASUREMENT ON REAL DATA)"""
selected_phd_opinion  = np.random.normal(true_stress, std_normal_dist * 0.1, (true_stress.size ))
```

MCMC set up:

- how many walkers (aka hobbits)
- burn in steps (training for the long hike to Mordor)
- Steps (to Mordor)
- Initial position of the walkers (Hobbiton, Shire, NW13 0LA)
- Sampler: magically tells the walkers where to go in the parameter space (aka Gandalf)

Result: CHAIN (it has kept track of the hobbits wandering around)

Again in python (1):

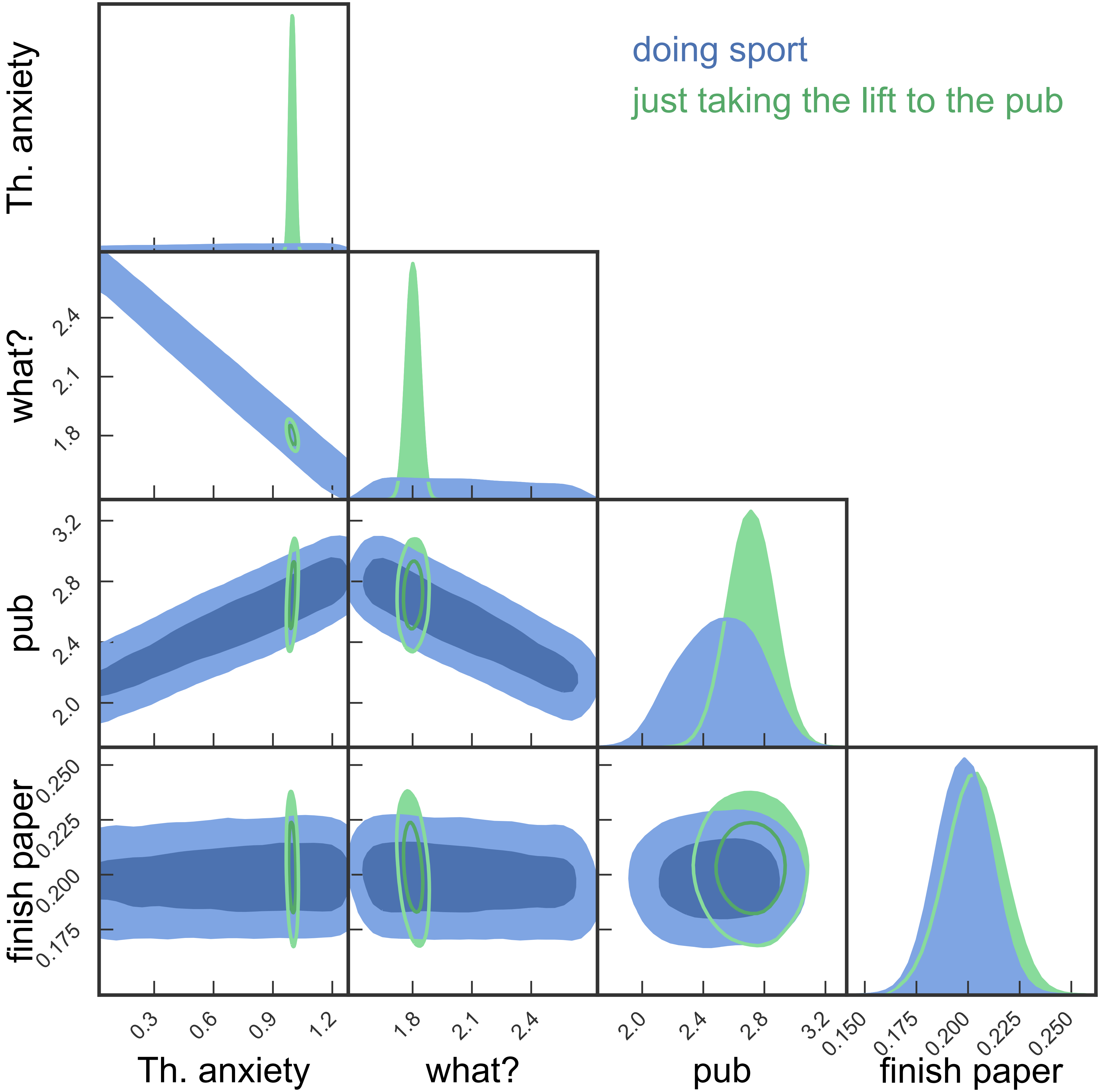
UCL

```python
""" walkers """
n_hobbits       = 200

""" burn_in """
warm_up_steps   = 200

""" steps tracked in the chain """
steps_to_mordor = 1000

""" run in parallel using MPI """
mpi_or_not      = False

""" initial guess for the maximum likelihood parameters """
shire_perimeter = np.array([[0.8,1.2], [1.5,2.5], [2.0,3.5], [0.0,0.6], [0.3,1.0]])
if doing_sport == False:
    shire_perimeter = np.array([[0.8,1.2], [1.5,2.5], [2.0,3.5], [0.0,0.6]])

""" starting point of walkers based on the initial guess """
hobbiton        = np.random.rand(par_num * n_hobbits).reshape((par_num,n_hobbits))

for i in range(par_num):
    hobbiton[i] =  shire_perimeter[i,0] + (shire_perimeter[i,1] - shire_perimeter[i,0]) * hobbiton[i]

hobbiton        = hobbiton.T

""" additional parameters to pass to the model function, including coordinates """
params          = time_ar, weather, middle_earth, doing_sport
```

Again in python (2):

UCL

```
if mpi_or_not == False:

    """ initialisate the MAGIC sampler """
    gandalf = emcee.EnsembleSampler(n_hobbits, par_num, lnprob_f,
                                    args=[selected_phd_opinion, params, np.linalg.inv(cov_matrix)], threads = 1)

    "burn in"
    pos, prob, state = gandalf.run_mcmc(hobbiton, warm_up_steps)
    gandalf.reset()

    "run mcmc"
    gandalf.run_mcmc(pos, steps_to_mordor)

    "reshape"
    samples = gandalf.chain[:, :, :].reshape((-1, par_num))

    "intervals"
    mcmc_list = []
    mcmc_list = map(lambda v: (v[1], v[2] - v[1], v[1] - v[0]),
                    zip(*np.percentile(samples, [16, 50, 84],
                                       axis=0)))
```



1-2D marginalised posterior distribution plot

References:

- https://github.com/davidegua/mcmc_example
- http://dfm.io/emcee/current/

Gualdi et al. in prep. (2018)