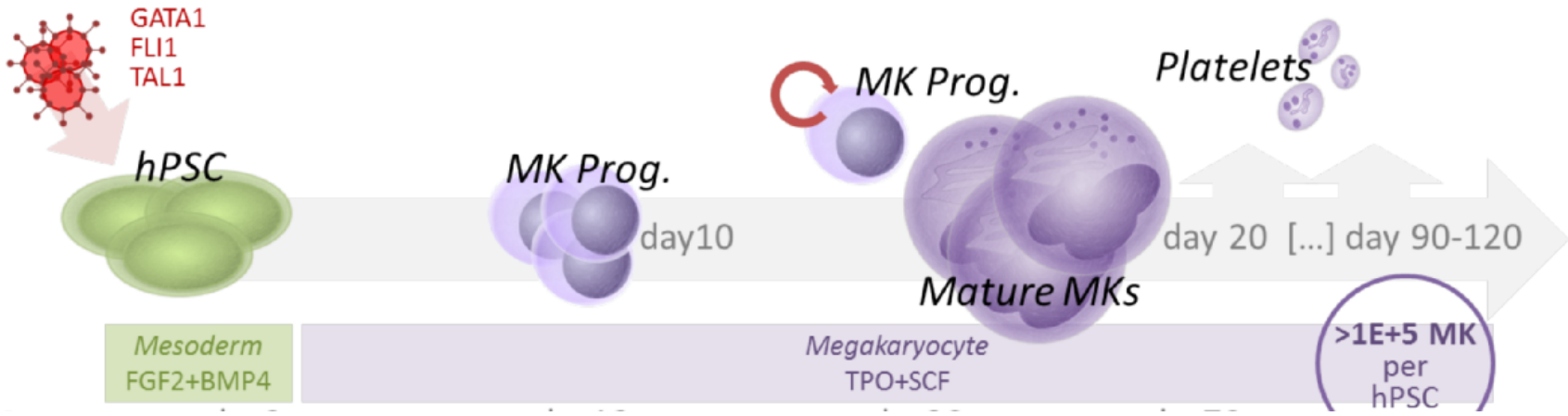
Improving protocols for cell differentiation through reinforcement learning

Lorenz Wernisch, John Reid, Cedric Ghevaert, Thomas Moreau

MRC-Biostatistics Unit and Cambridge Blood Centre
Cambridge, UK

7 June 2018

Stem cells to Megakaryocytes

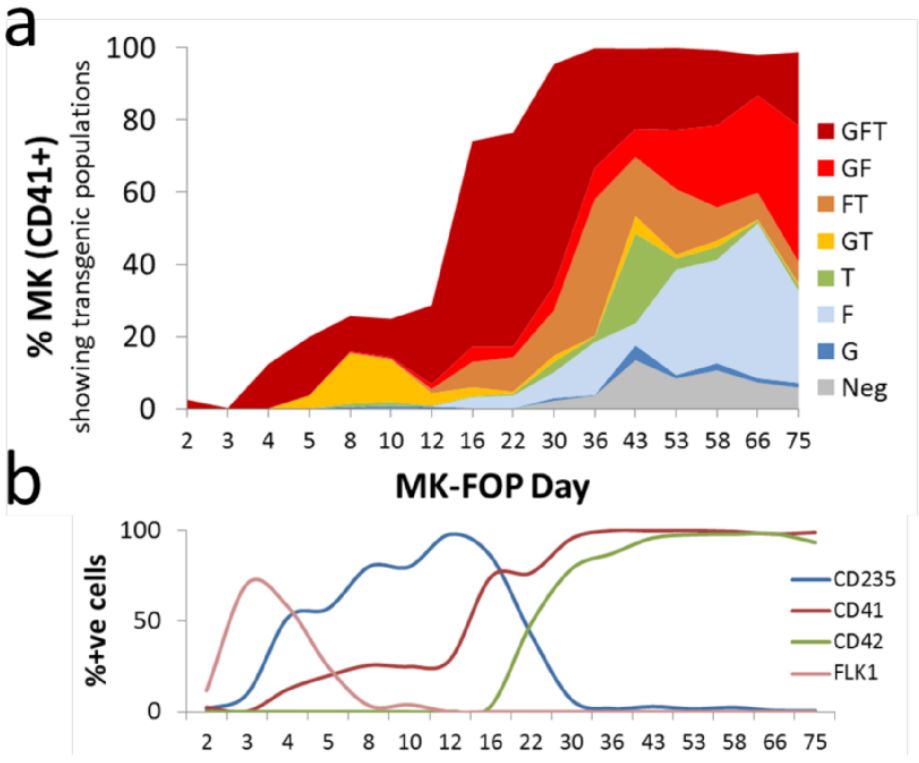


Human pluriopotent stem cells to Megakaryocytes via induction of genes GATA1, FLI1, TAL1

Increase yield and maturity of MK cells

Problem: When and how much induction is required, what other

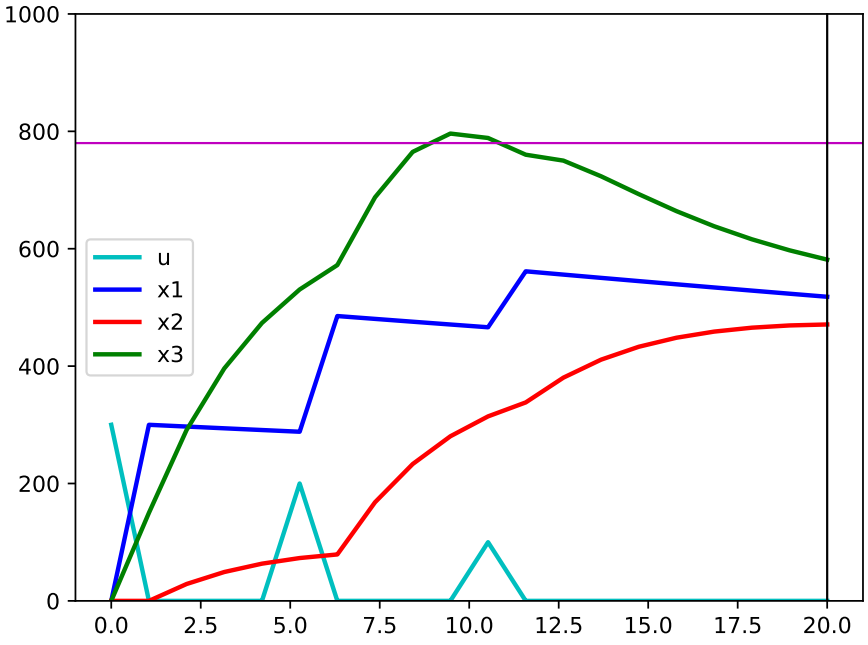
Flow cytometry measurements



Cell surface markers and gene transcription profile indicate maturity

Yield of MK cells from cell cultures

Toy abstraction of problem

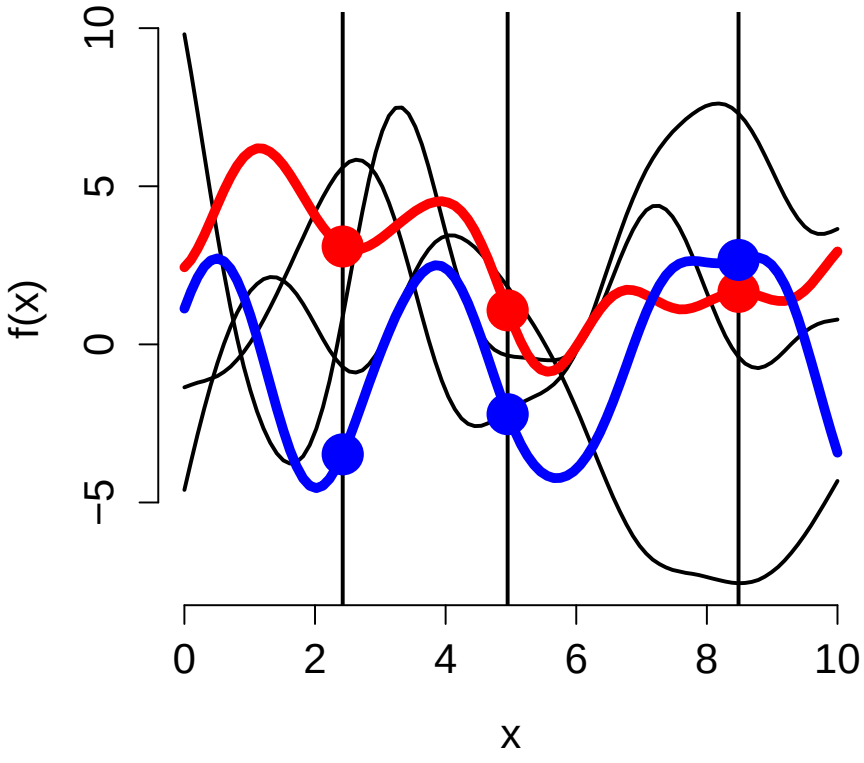


System of three variables $x_{1,t}, x_{2,t}, x_{3,t}$ measured daily over 20 days

We control input u_t to push $x_{3,20}$ to a target value on day 20

$$\begin{pmatrix} x_{1,t} \\ x_{2,t} \\ x_{3,t} \end{pmatrix} = \begin{pmatrix} x_{1,t-1} \\ x_{2,t-1} \\ x_{3,t-1} \end{pmatrix} + \begin{pmatrix} f_1(x_{1,t-1}, x_{2,t-1}, x_{3,t-1}, u_{t-1}) \\ f_2(x_{1,t-1}, x_{2,t-1}, x_{3,t-1}, u_{t-1}) \\ f_3(x_{1,t-1}, x_{2,t-1}, x_{3,t-1}, u_{t-1}) \end{pmatrix}$$

Gaussian process prior



Family of functions via covariance K on input points x

$$y \sim N(0, K_{xx})$$

Prediction for x^* from (x, y)

$$y^* \sim N(K_{x^*x}K_{xx}^{-1}y, \Sigma)$$

$$\Sigma = K_{x^*x^*} - K_{x^*x}K_{xx}^{-1}K_{xx^*}$$

Gaussian $\mathrm{cov}(x, x^*) = \theta_1 \exp(-\theta_2(x - x^*)^2)$

Matern $\mathrm{cov}(x, x^*) = \theta_1(1 + \theta_2|x - x^*|)\exp(-\theta_2|x - x^*|)$

Dynamical system via Gaussian processes

Idea: use data to approximate the difference

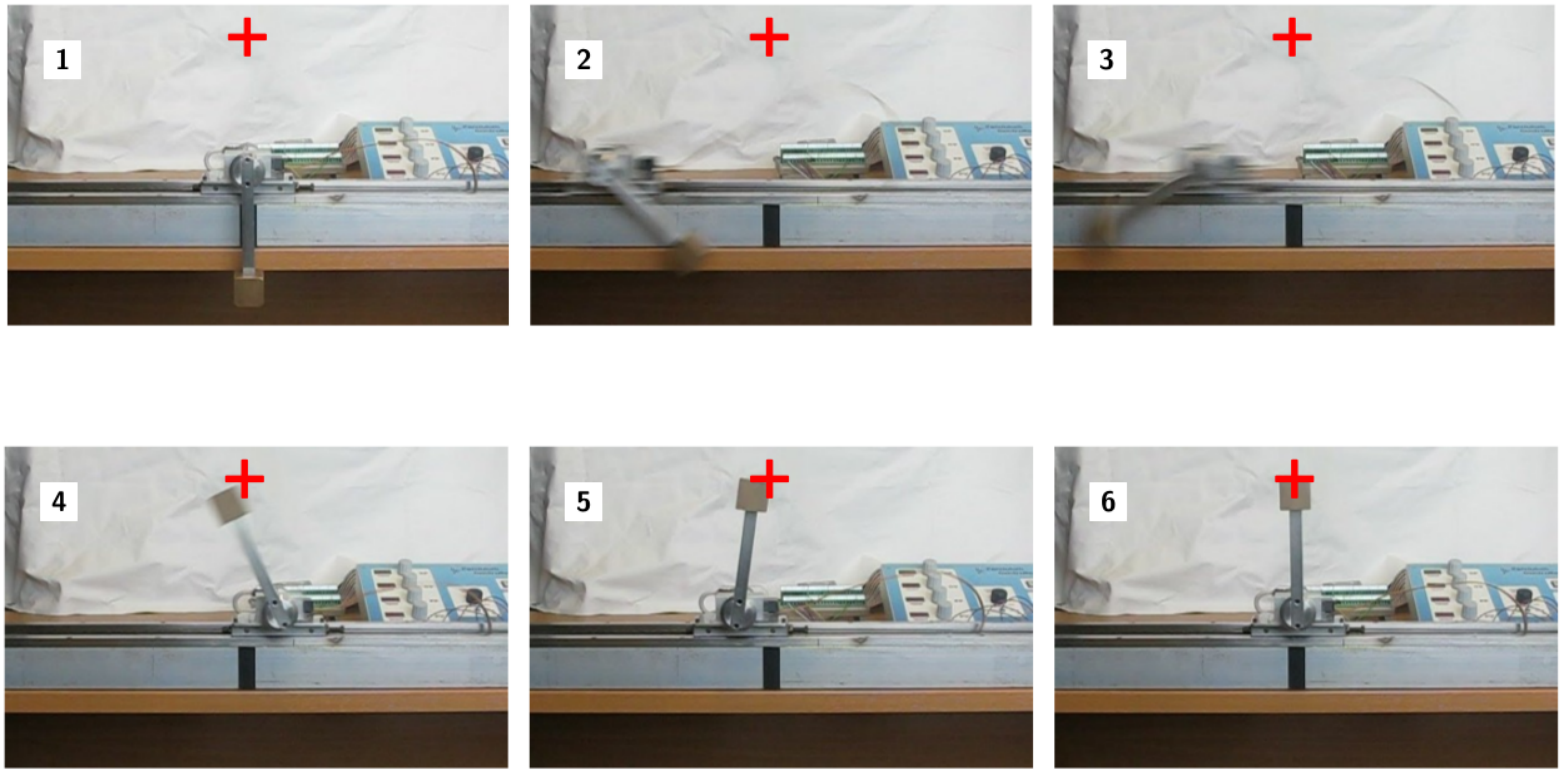
$$x_{i,t} - x_{i,t-1} = f_i(x_{1,t-1}, \ldots, x_{d,t-1}, u_{t-1})$$

via d GPs $f_i \sim GP(\mu(x,u), \Sigma(x,u))$, $i = 1, \ldots, d$

Iterative cycle for *optimising control* towards target:

- Approximate response x to u by GPs
- Optimize control input u towards target using GP approximation
- Acquire new data applying control to real system

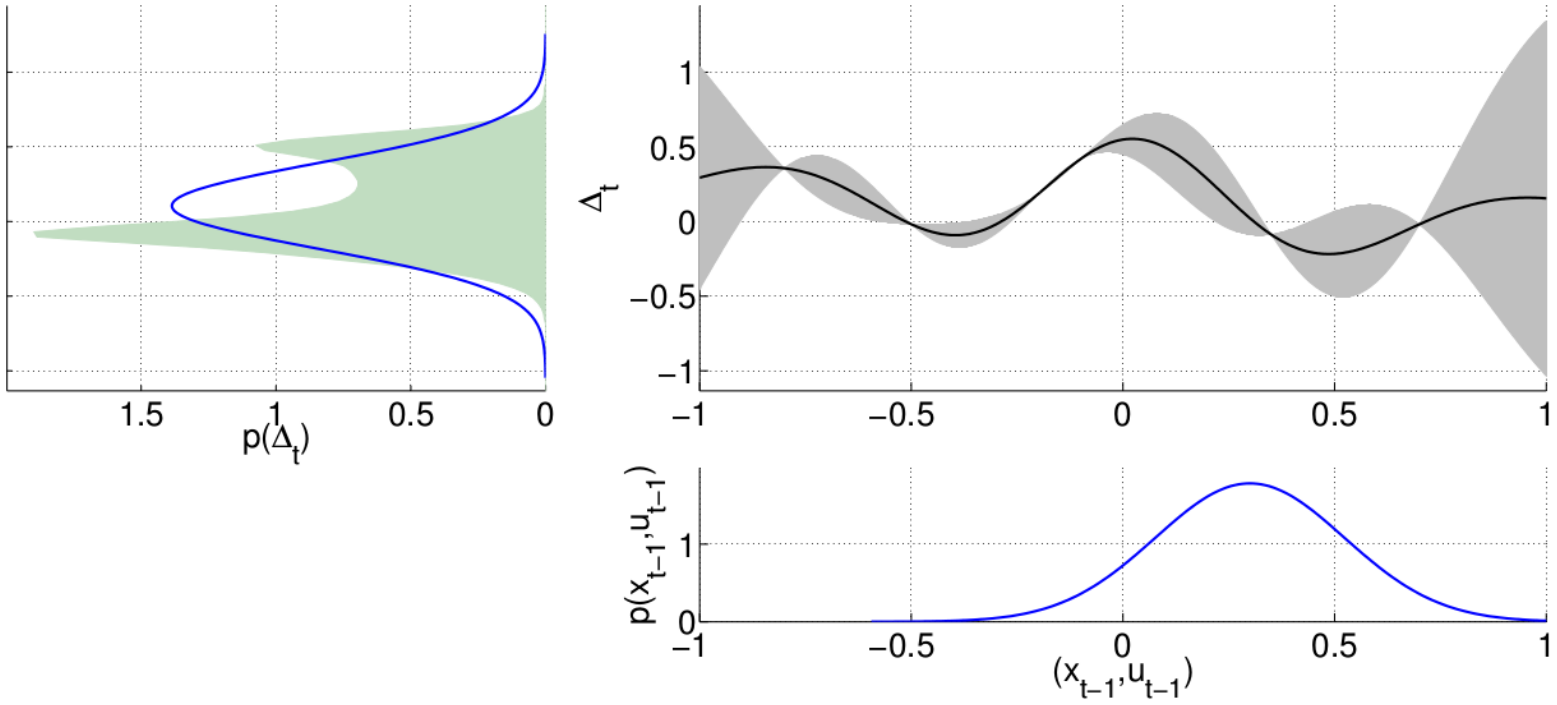
Pilco: probabilistic learning of control



Learn to move cart left/right to swing up pole

Deisenroth and Rasmussen, 2011: PILCO: A Model-Based and Data-Efficient Approach to Policy Search (Probabilistic Inference for Learning COntrol)

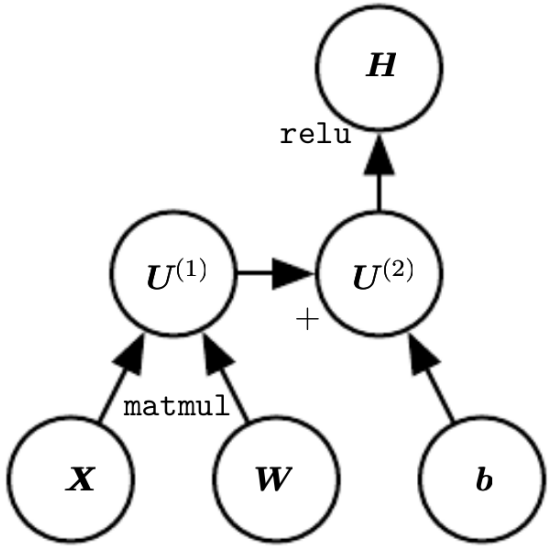
Transmitting uncertainty



(Gaussian) uncertainty in *inputs* results in (non-Gaussian) distribution in output

Pilco: approximate by Gaussian via moment matching

Alternative: dataflow graphs



Tensorflow works with *stateful dataflow graphs*

Data flow between nodes in a directed graph

Nodes represent eg: arithmetic operations, control clauses (if else), matrix manipulations, random number generators

Automatic differentiation

Optimize u using GP approximation

Minimize cost $c(u)$, eg for trajectory $x_3(u_0, \ldots, u_{19})$

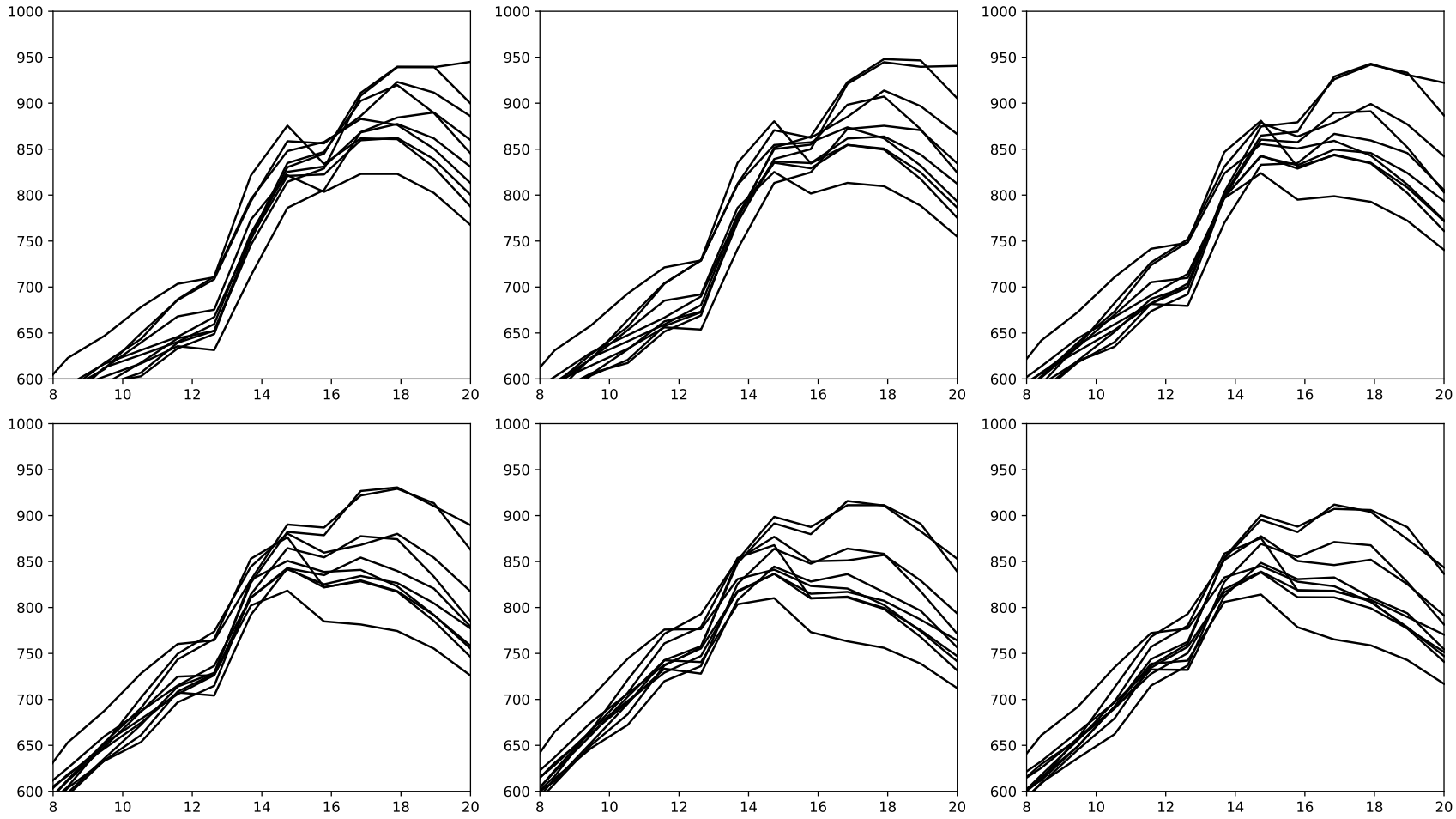
$$c(u) = (x_{3,20}(u_0, \ldots, u_{19}) - x_{\text{target}})^2 + \lambda \sum_t |u_t|$$

How to define trajectory using GPs?

Bad idea: use means of GP for $f_i(x_{t-1}, u_{t-1})$

Better idea: sample several *random* trajectories using GP uncertainty and optimise eg mean

Random trajectories



Reparameterisation trick

Sample m trajectories $x^{(k)} \sim p_{\mathrm{GP}}(\cdot \mid u)$ for input u according to GPs

Compute $C(u) = \frac{1}{m} \sum_k c(x^{(k)})$ using sample

No dependency on u: gradient $\nabla_u C(u) = 0$!?

Solution: reparameterise $x^{(k)} = g(u, \epsilon_k)$, with ϵ_k from a standard distribution, so that $x^{(k)} \sim p_{\mathrm{GP}}(\cdot \mid u)$

$$\nabla_u C(u) = \frac{1}{m} \sum_k \nabla_u c(g(u, \epsilon_k))$$

for a fixed sample ϵ_k

Reparameterisation for Gaussian

$p_N(z \mid \mu(u), \Sigma(u))$

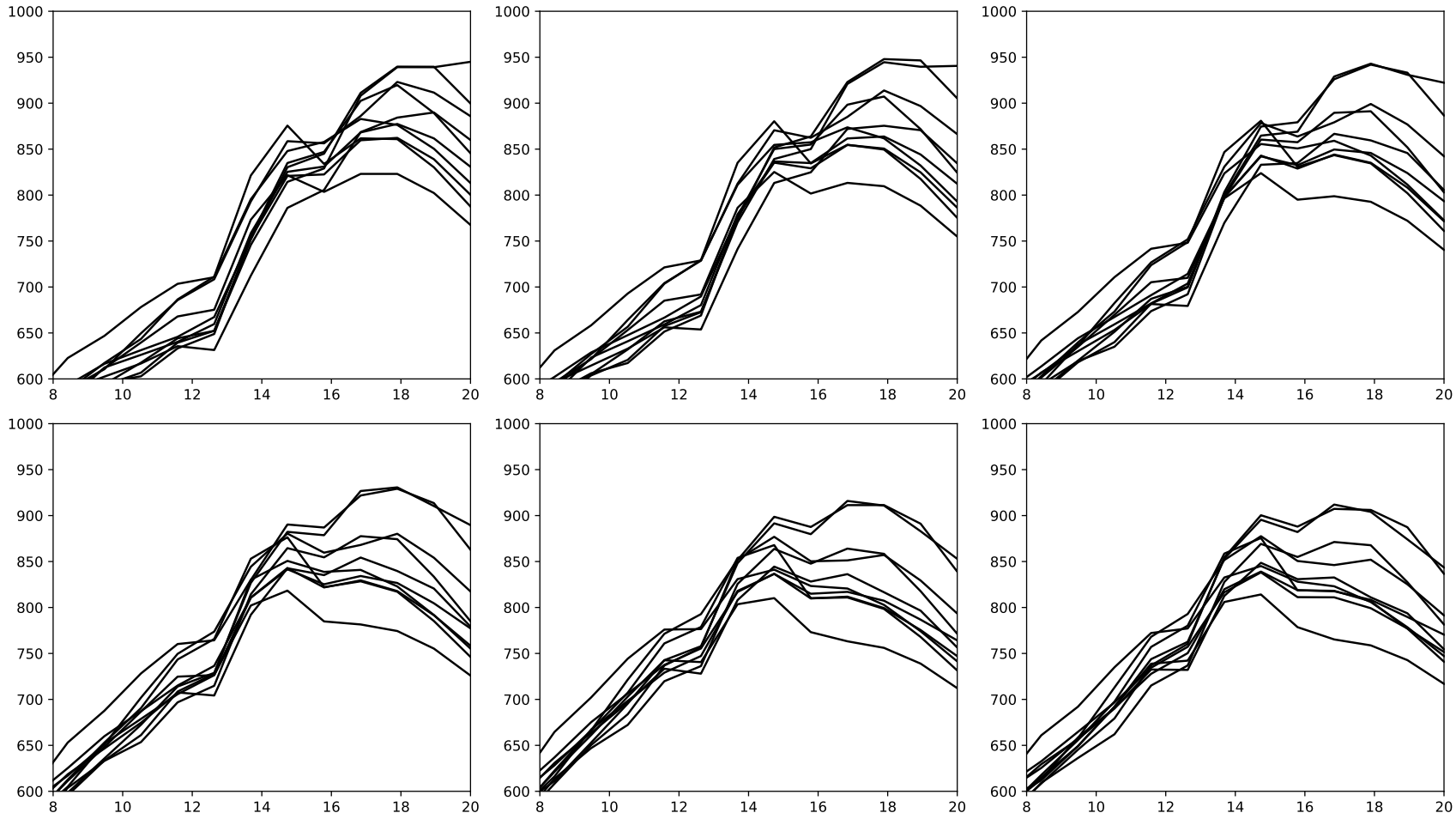
Choleski factorisation $\Sigma(u) = L(u)L(u)^T$

With $\epsilon \sim N(0, I)$

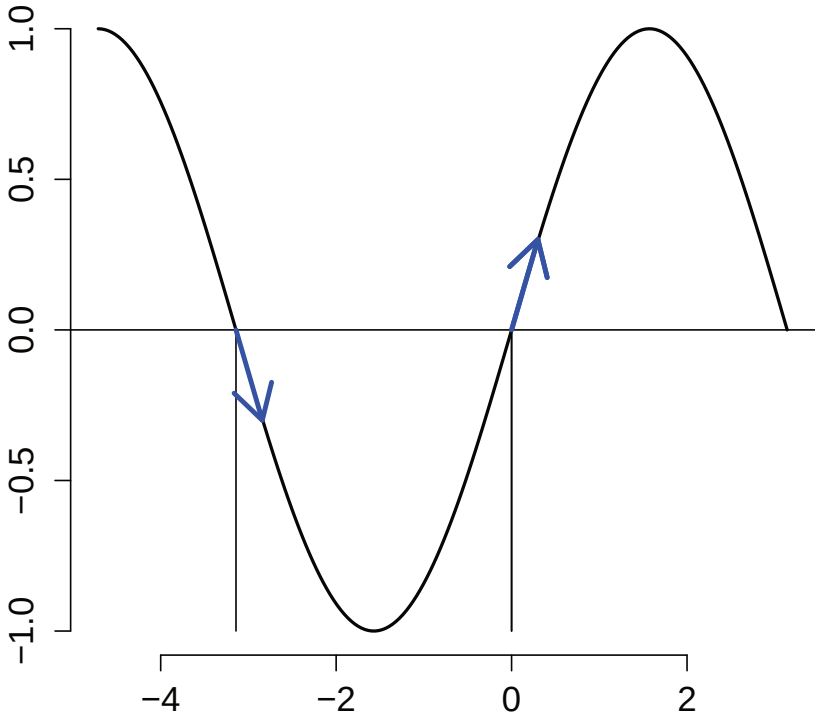
$$z(u) = g(\epsilon, \mu(u), \Sigma(u)) = \mu(u) + C(u)\epsilon$$

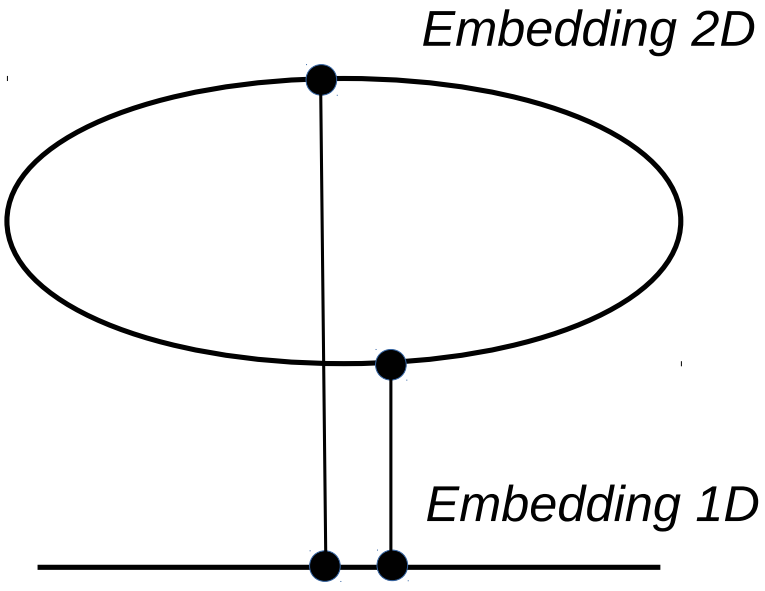
$\mu(u), C(u)$ from GPs trained on data and test input u

Optimising with reparameterization trick



Key model parameter: latent dimension



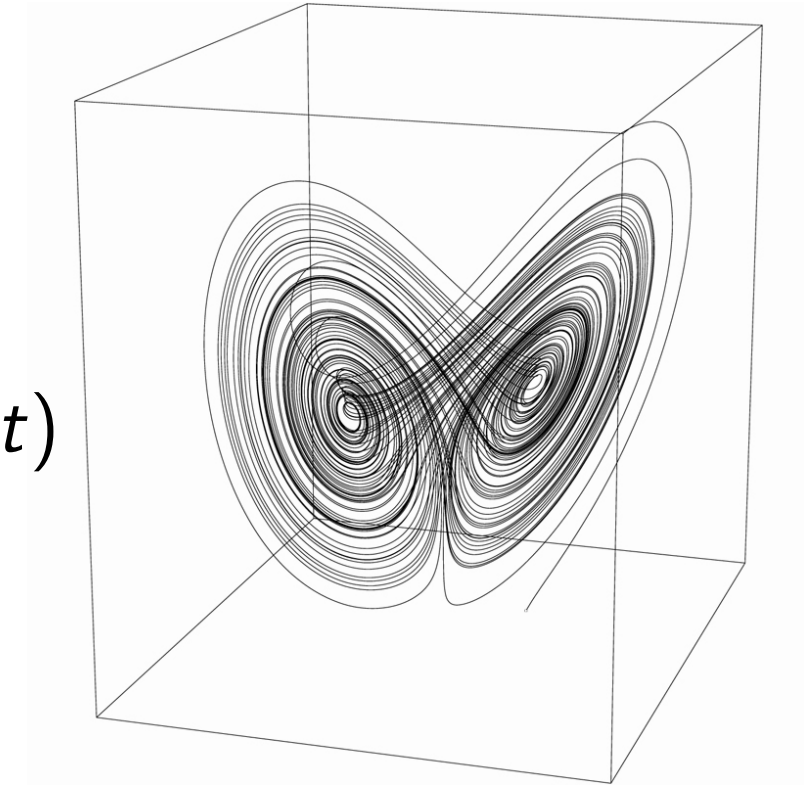


$\dot{y} = f(y)$ or $x_t = x_{t-1} + f(t_{t-1})$ not representable in 1D:

Identical y mapped to different $f(y)$

Lorenz attractor

$$\dot{x}(t) = 10(y(t) - x(t))$$
$$\dot{y}(t) = x(t)1(28 - z(t)) - y(t)$$
$$\dot{z}(t) = x(t)y(t) - 8/3z(t)$$



Trajectories end up on "strange attractor" of (noninteger) fractal dimension

Takens' theorem: "Lag embedding" of any one of x,y,z is diffeomorphic to attractor

Takens' theorem

Observe one variable $y(1), y(2), y(3), \ldots, y(L)$ from dynamical system

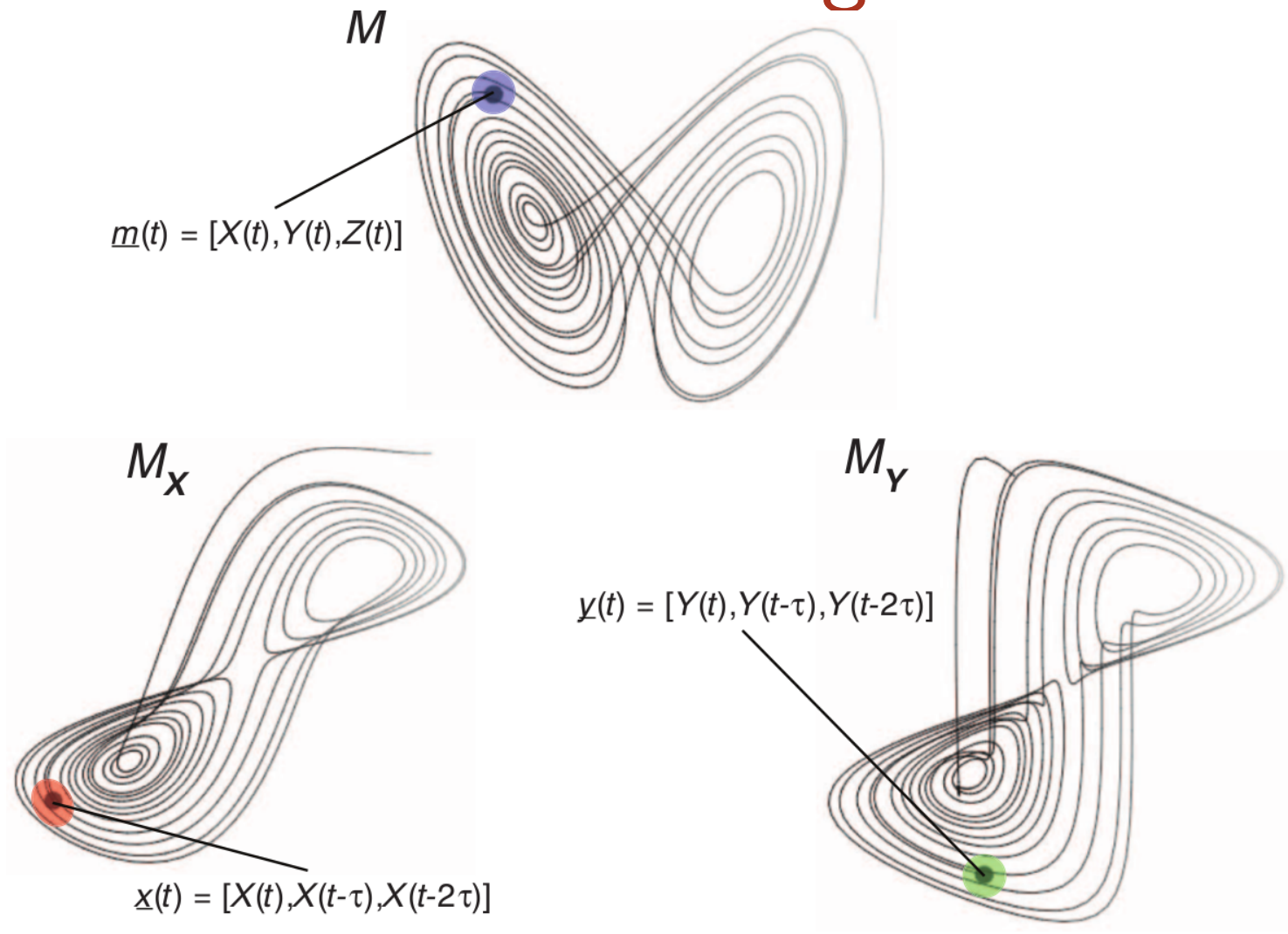
Find lag d with (eg by small autocorrelation)

Choose m and form delay embedding (eg $d = 2$, $m = 3$):

$$\begin{pmatrix} y(1) \\ y(3) \\ y(5) \end{pmatrix}, \begin{pmatrix} y(2) \\ y(4) \\ y(6) \end{pmatrix}, \begin{pmatrix} y(3) \\ y(5) \\ y(7) \end{pmatrix}, \ldots$$

Choose m with few "false" neighbors (monitor distance increase when adding $m + 1$st expansion)

Lorenz attractor embeddings



Sugihara et al, Detecting Causality in Complex Ecosystems, Science 2012

Rules of the game

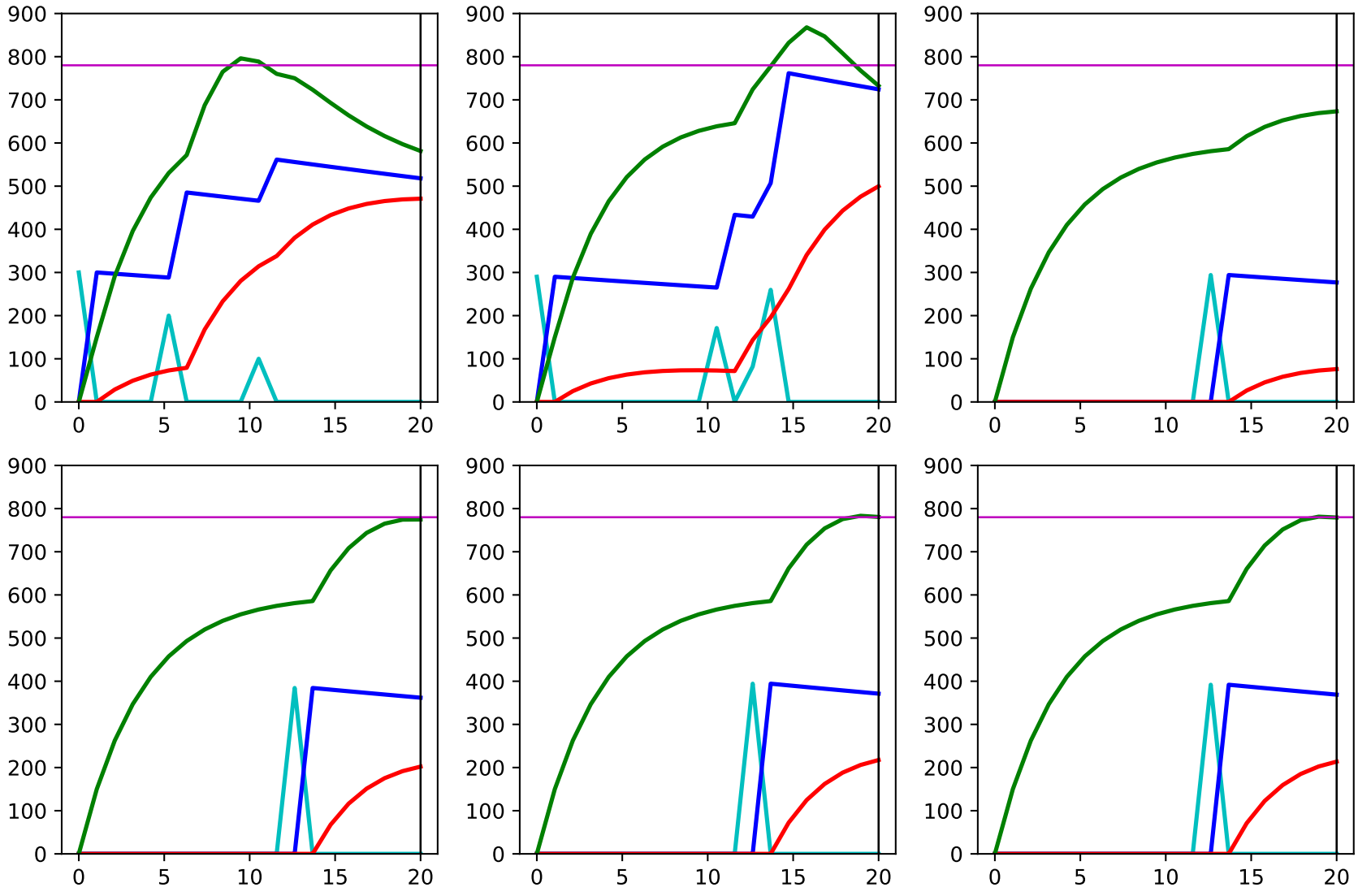
Given: black box dynamical system, cost function $c(x(u))$ for input u to the system

Choose some initial input $u = (u_0, \ldots, u_{T-1})$, iterate

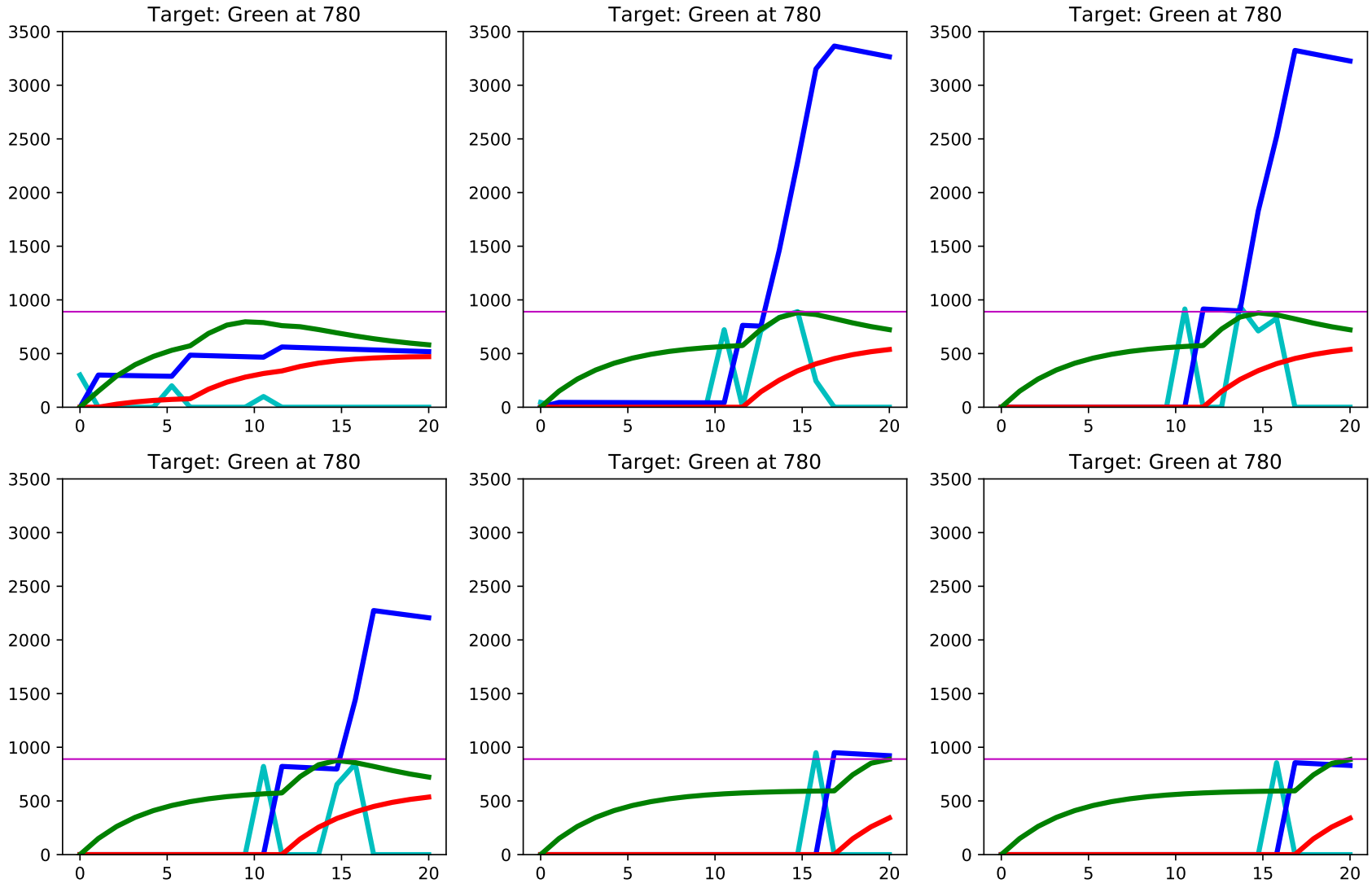
- Obtain $x_1, \ldots, x_t$ for input u into black box system
- Construct a model of the system and optimize u to achieve minimum $c(x(u))$

Can the true minimum of $c(x(u))$ be achieved? How many iterations?

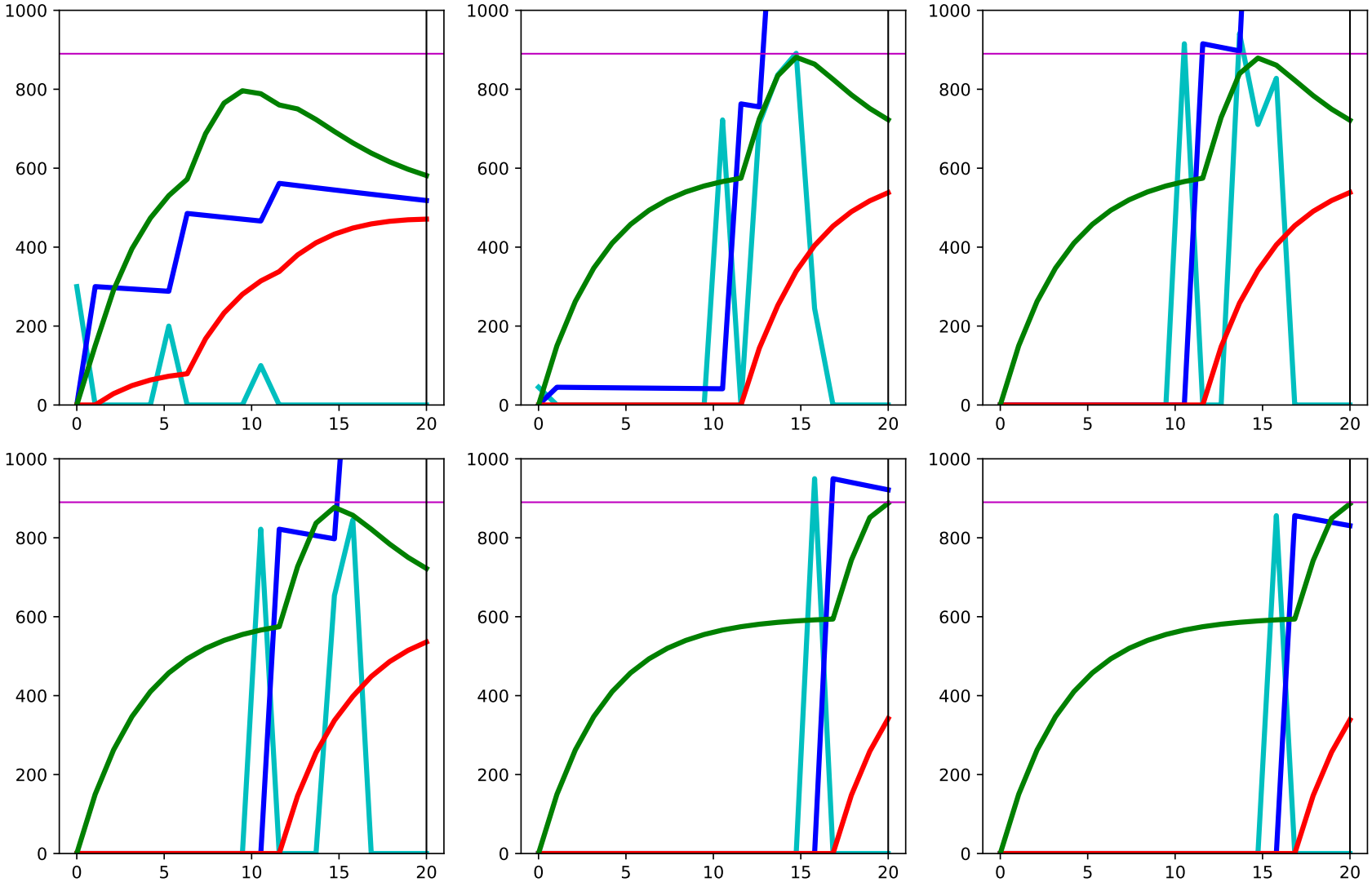
Aim: Green at 780 with few inputs



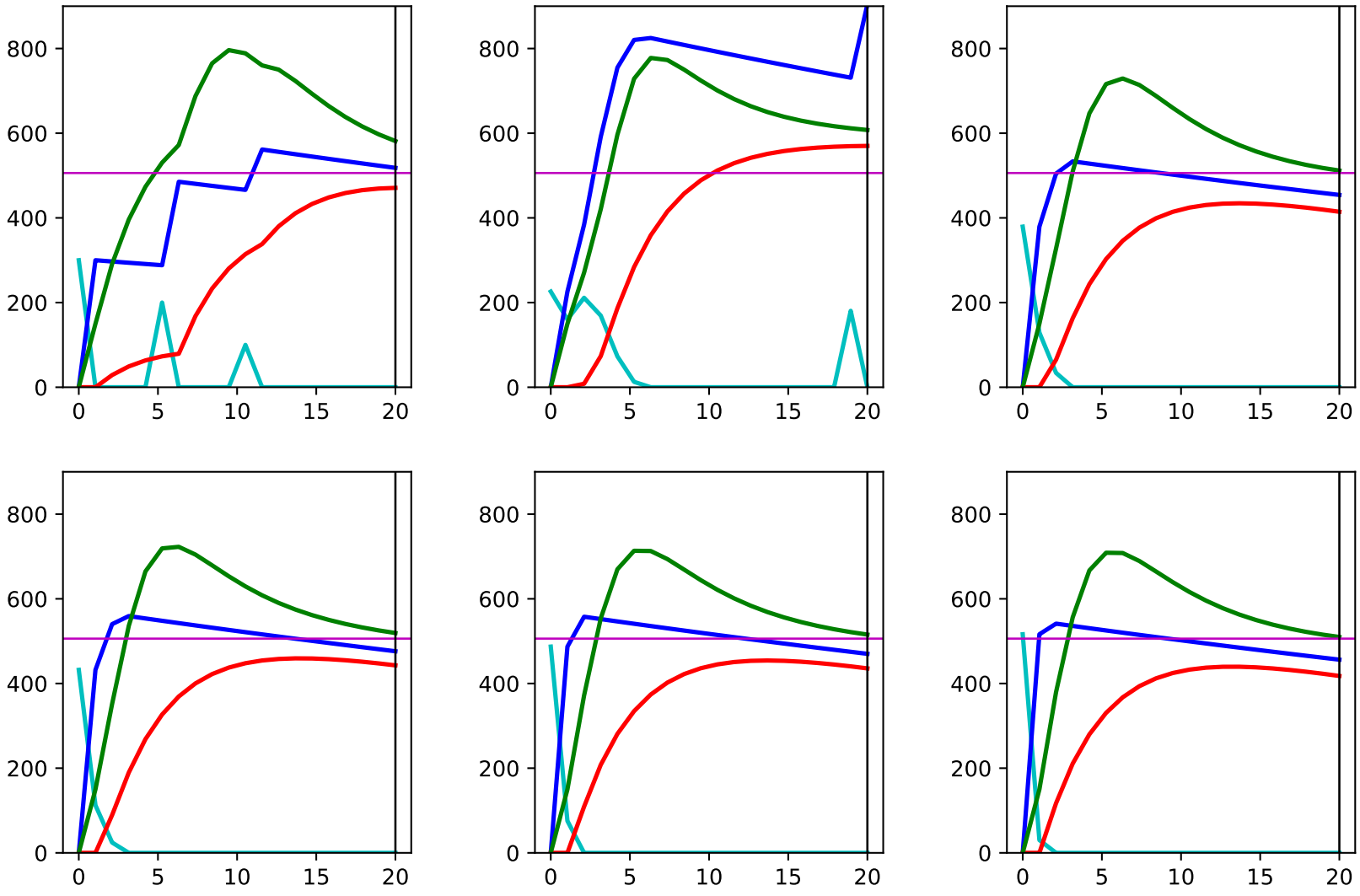
Aim: Green at maximum with few inputs



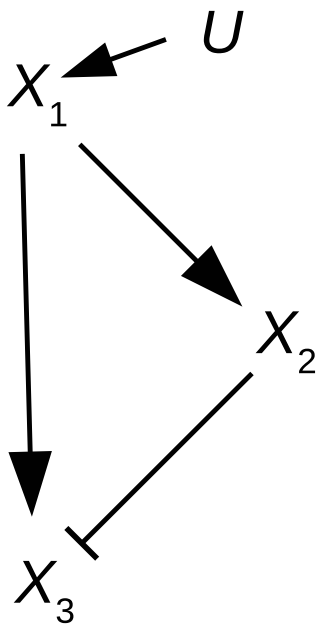
Zoom in: Green at maximum

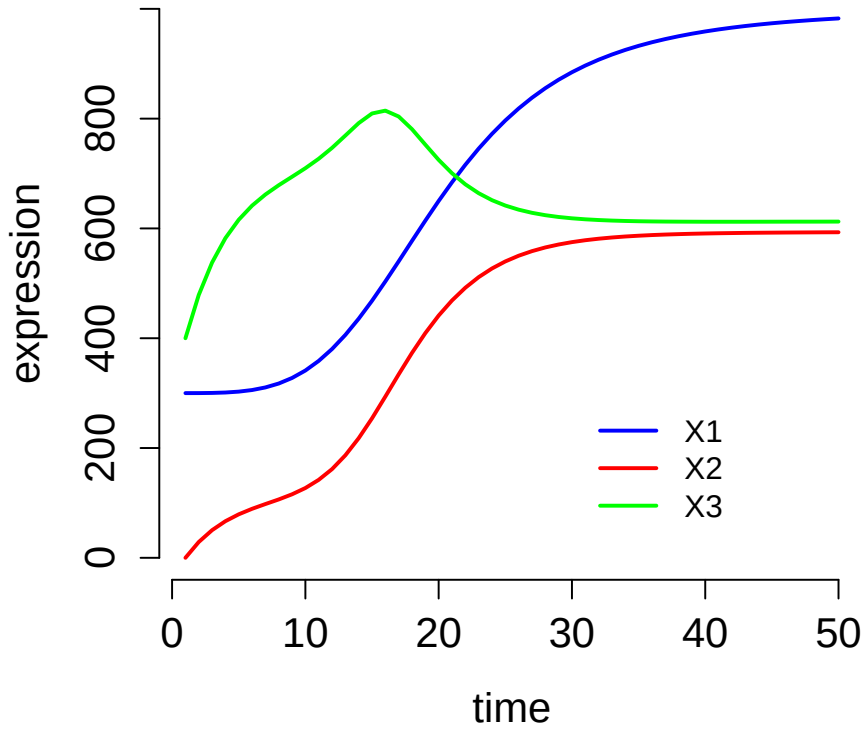


Aim: Green at minimum with few inputs



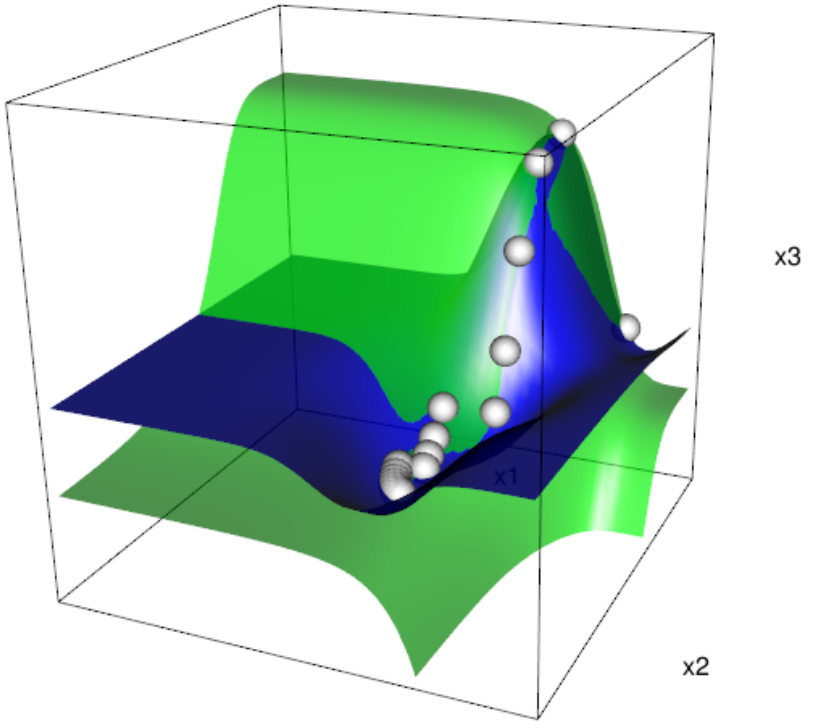
Feedforward loop

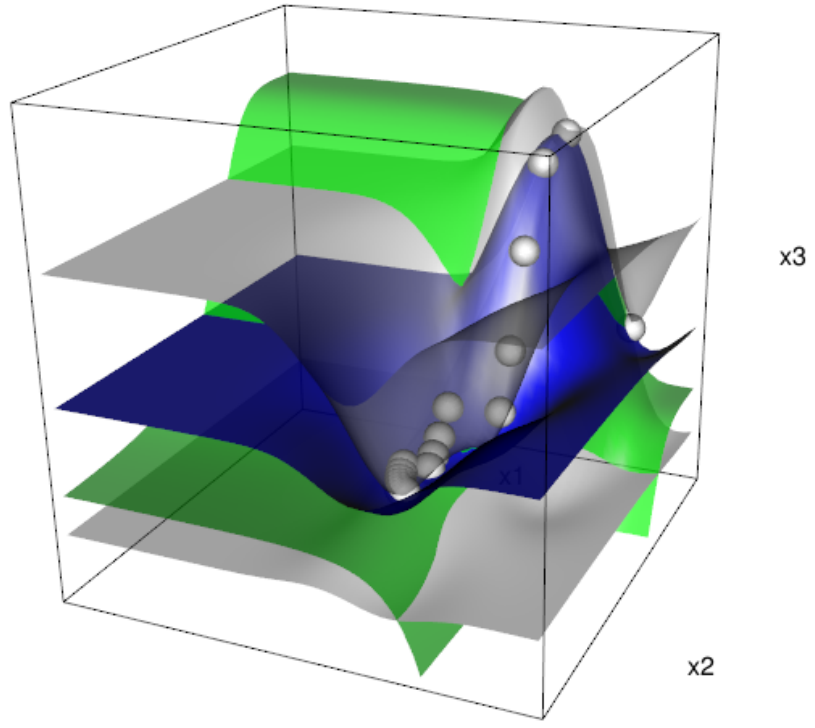




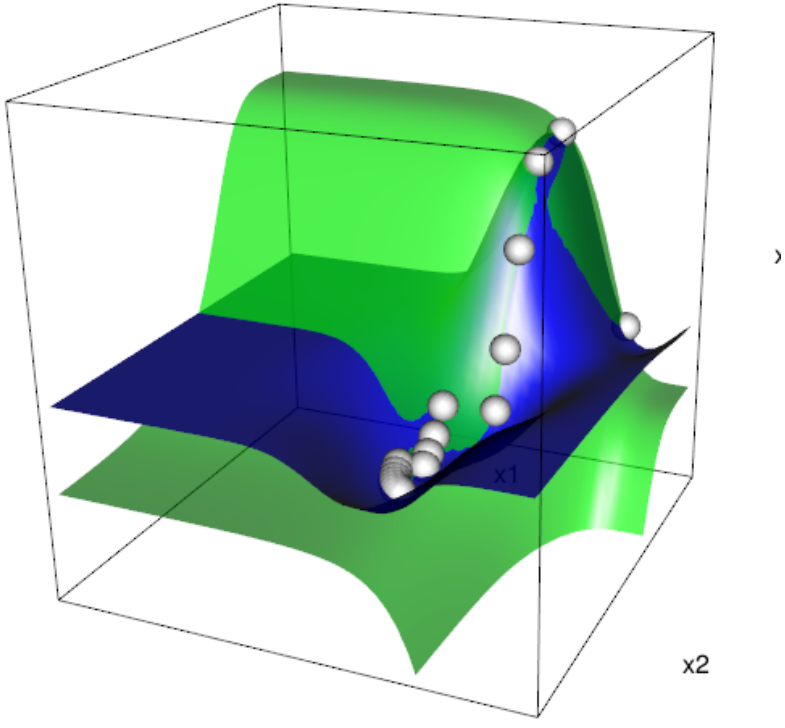
$$X_1(t) = (1-\lambda_1)X_1(t-1) + U_{\text{activate}}(t)$$
$$X_2(t) = (1-\lambda_2)X_2(t-1) + h^+(X_1(t-1))$$
$$X_3(t) = (1-\lambda_3)X_3(t-1) + h^+(X_1(t-1)) + h^-(X_2(t-1))$$

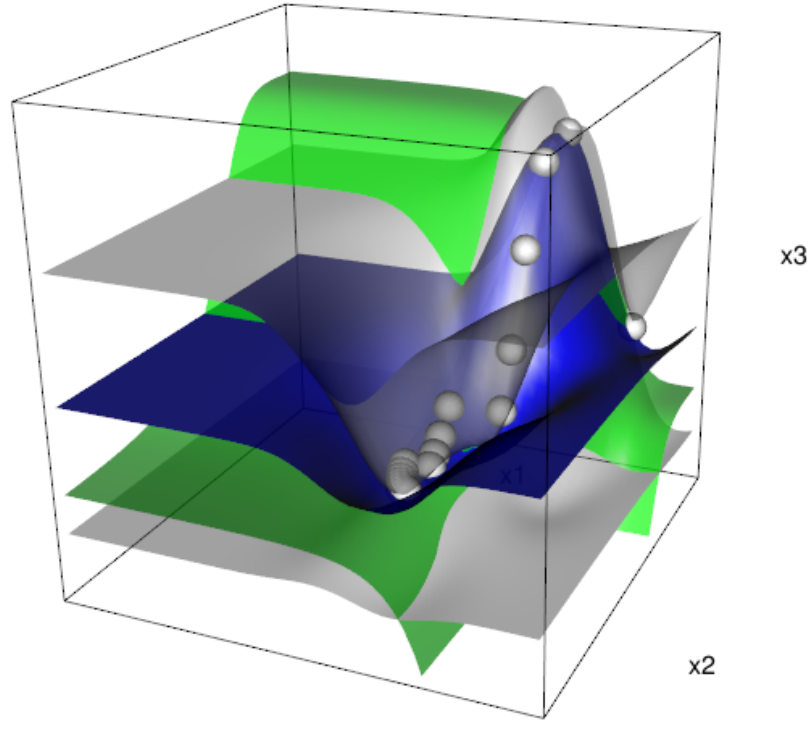
GP for $f_3(x_1, x_2)$ initial experiment



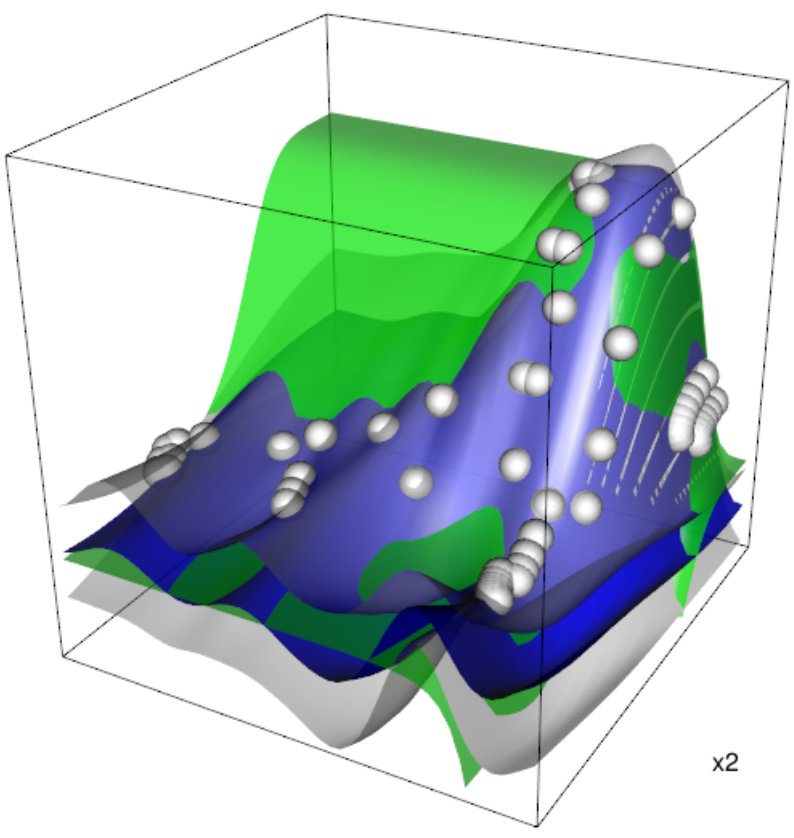


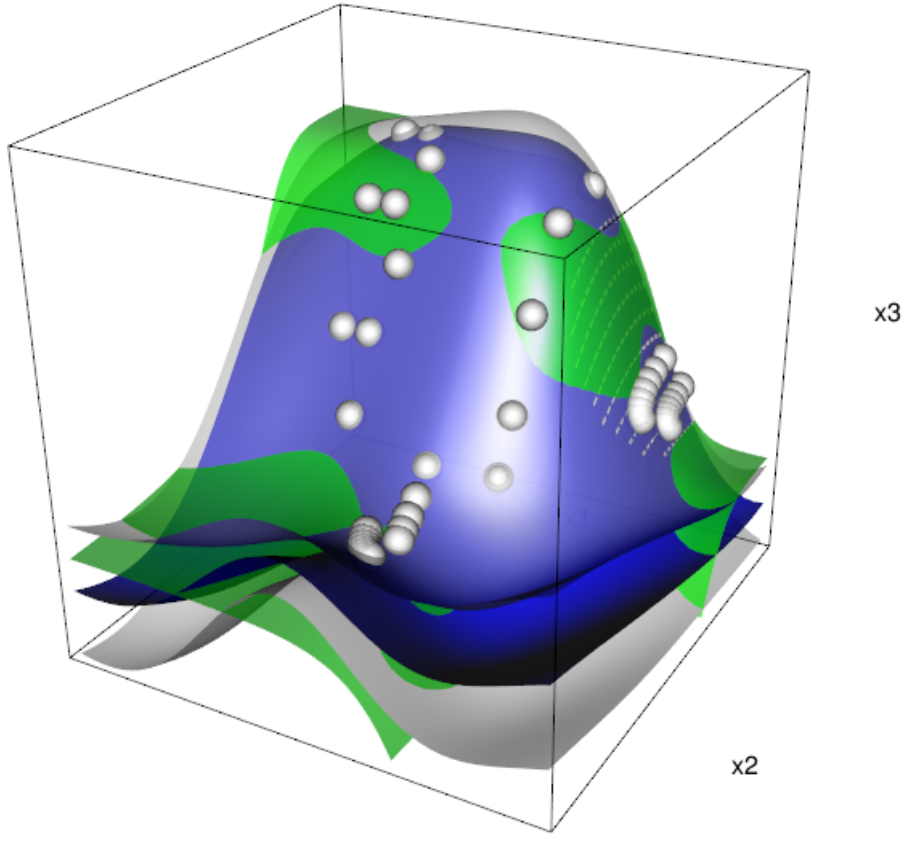
GP for $f_3(x_1, x_2)$ initial experiment



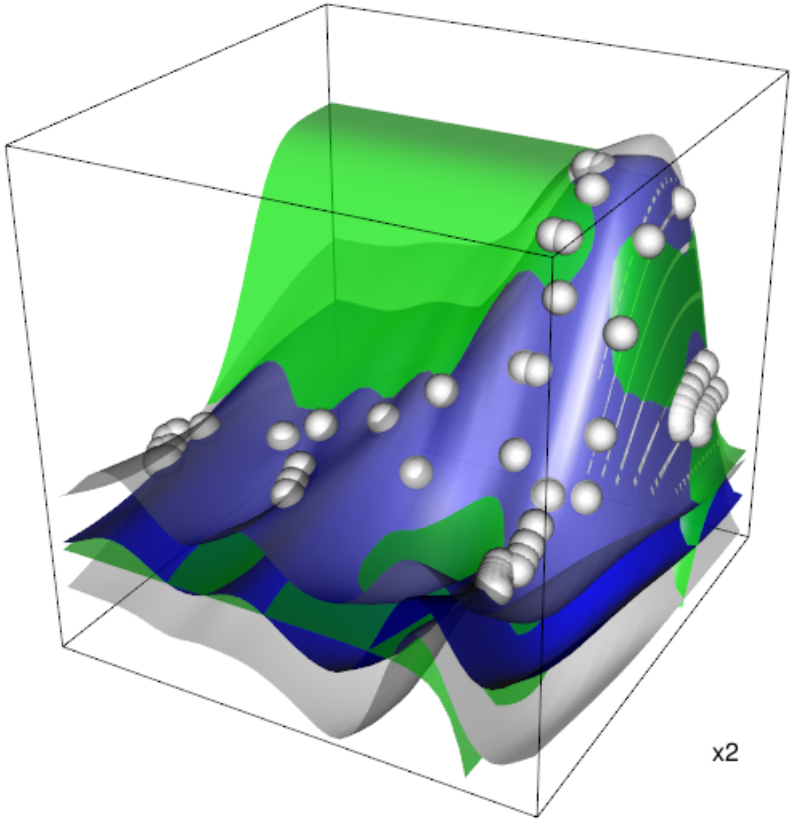


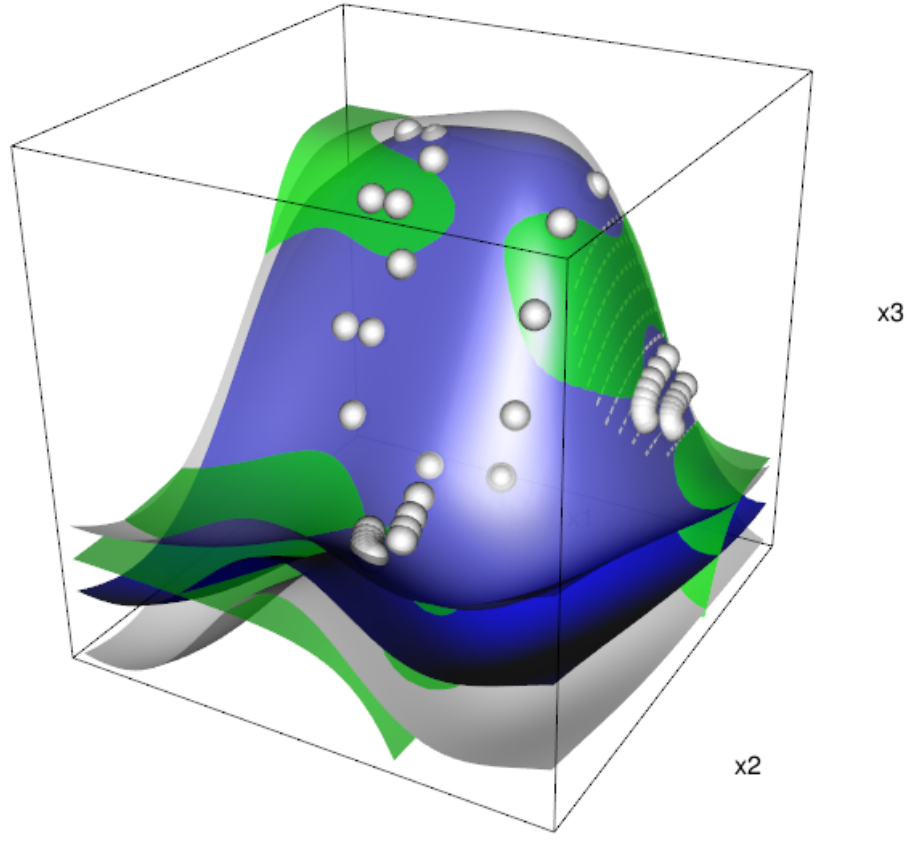
After four maximization experiments



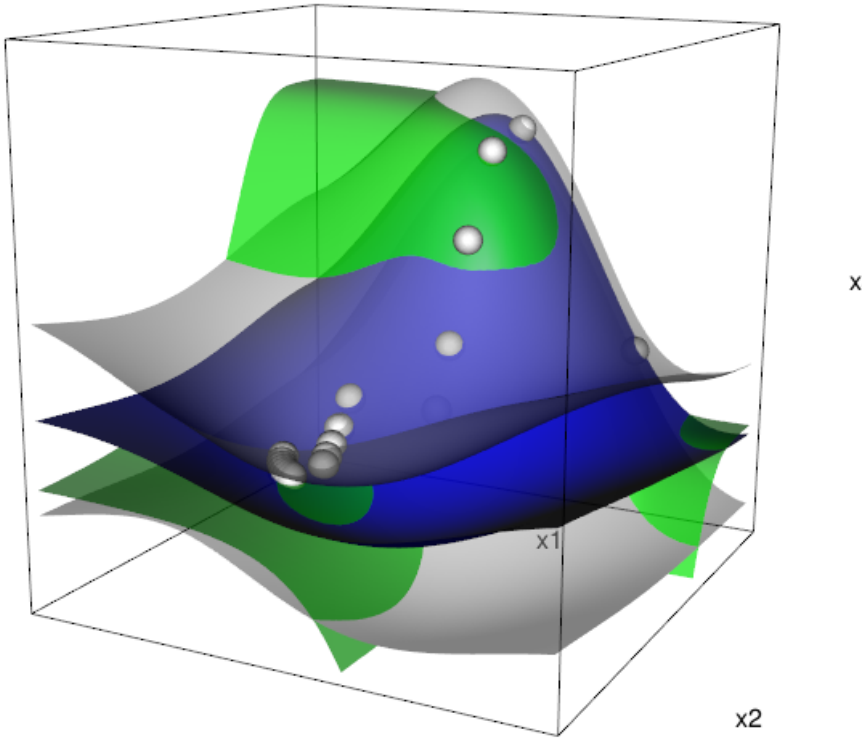


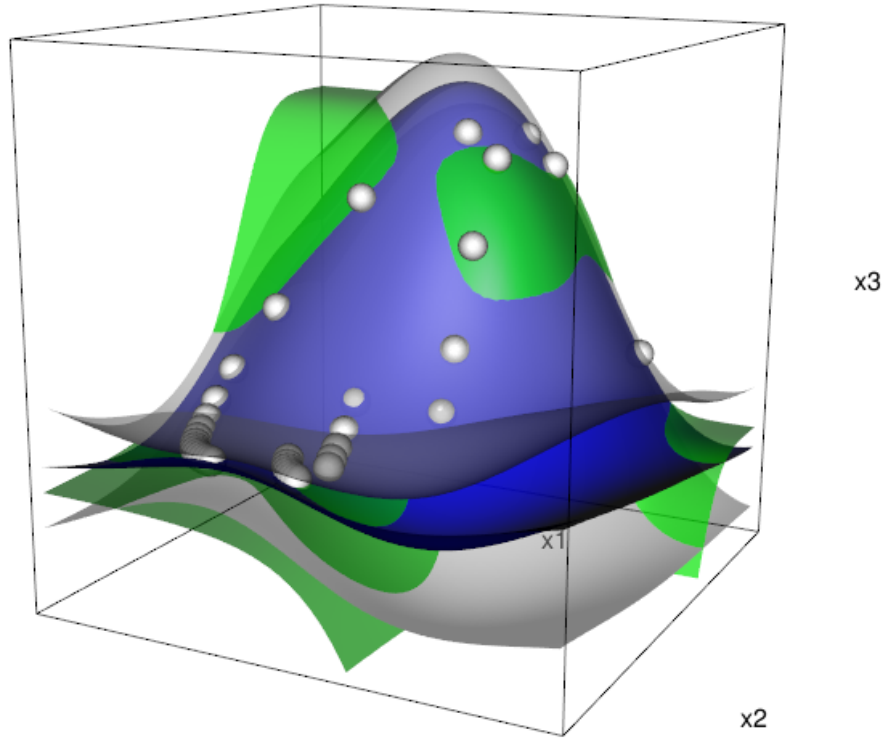
After four maximization experiments



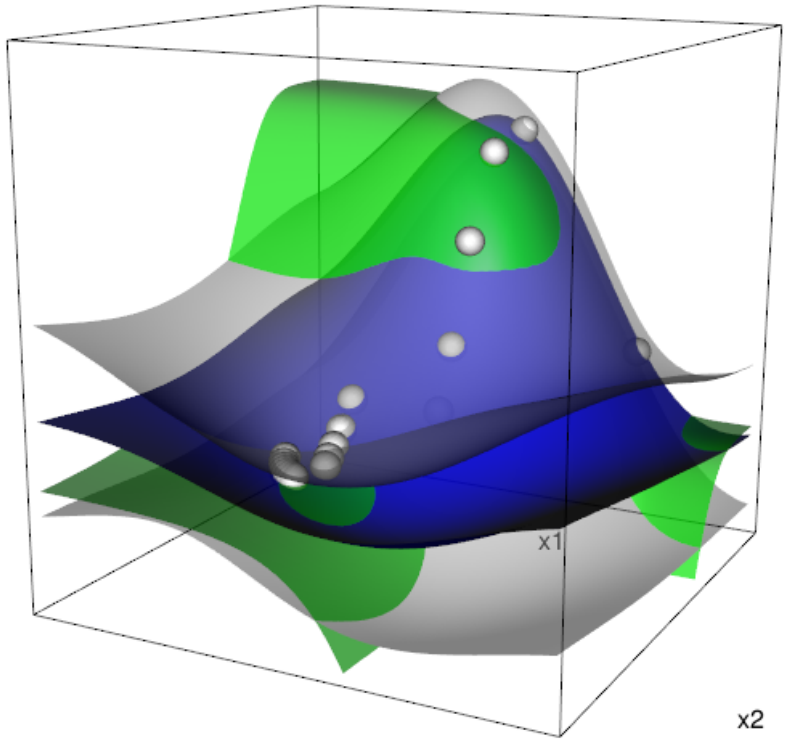


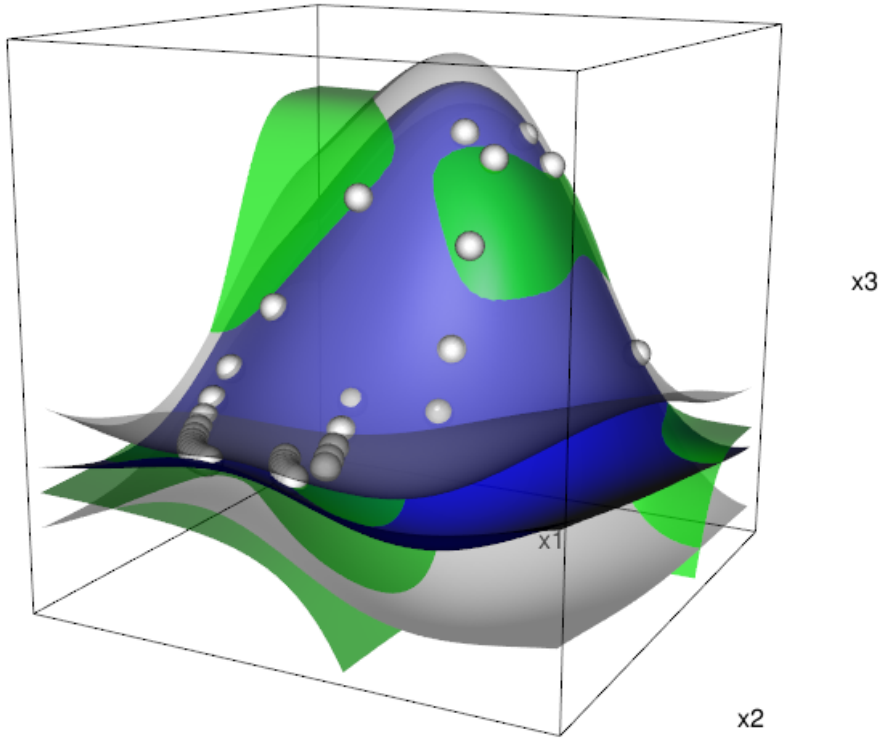
After two minimization experiments





After two minimisation experiments





Difference to Pilco

No Gaussian approximation via moment matching: trajectories are directly sampled from GP distribution

Tensorflow framework and reparameterisation trick for automatic gradient calculation and efficient optimisation

No need to restrict to special GP kernels (Gaussian), and to explicitly derive gradients

Extremely flexible regression modelling and cost functions

Afterthoughts

Taking uncertainty into account is essential: deterministic version (eg via GP mean function) does not work

Strong regularising effect of using expectation of cost

Converges surprisingly quickly

Dynamic control problems widely applicable

Tensorflow (or similar frameworks) enable straightforward, flexible implementation, can be easily adapted and expanded