COMP261 Lecture 22

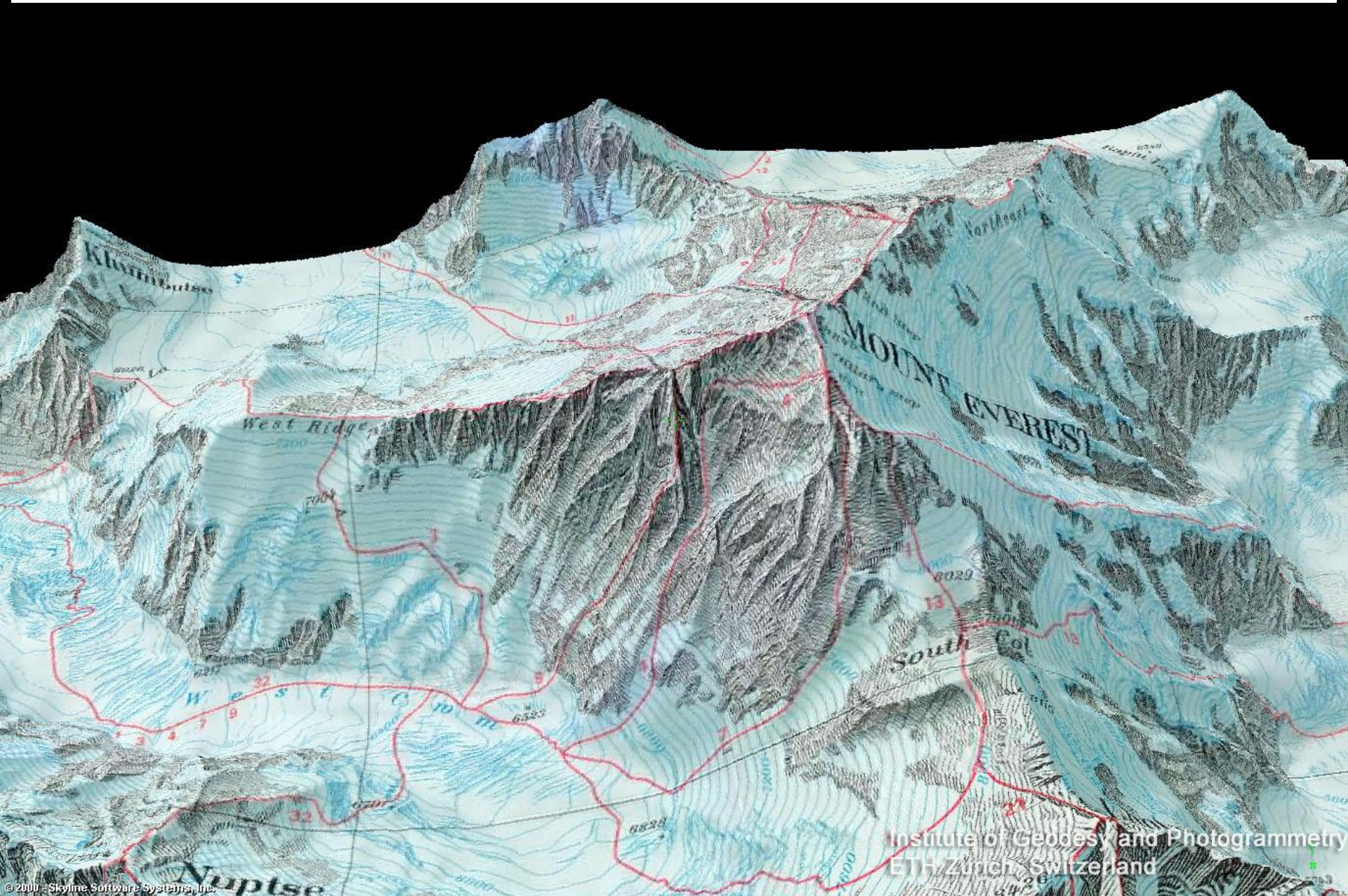
Marcus Frean

(Using Predictions 2)

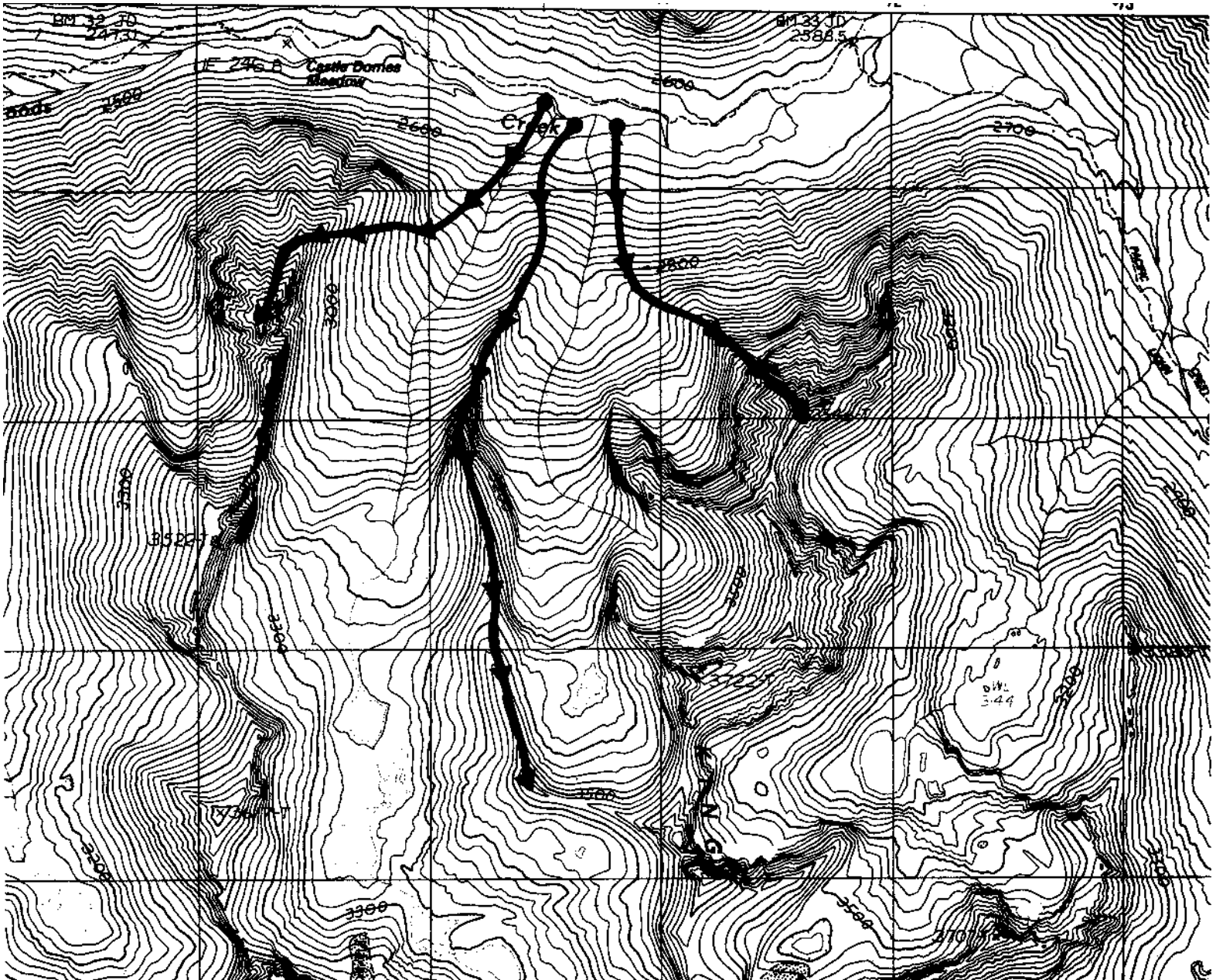
Optimization of an expensive function

Victoria
UNIVERSITY OF WELLINGTON
Te Whare Wānanga
o te Ūpoko o te Ika a Māui
CAPITAL CITY UNIVERSITY

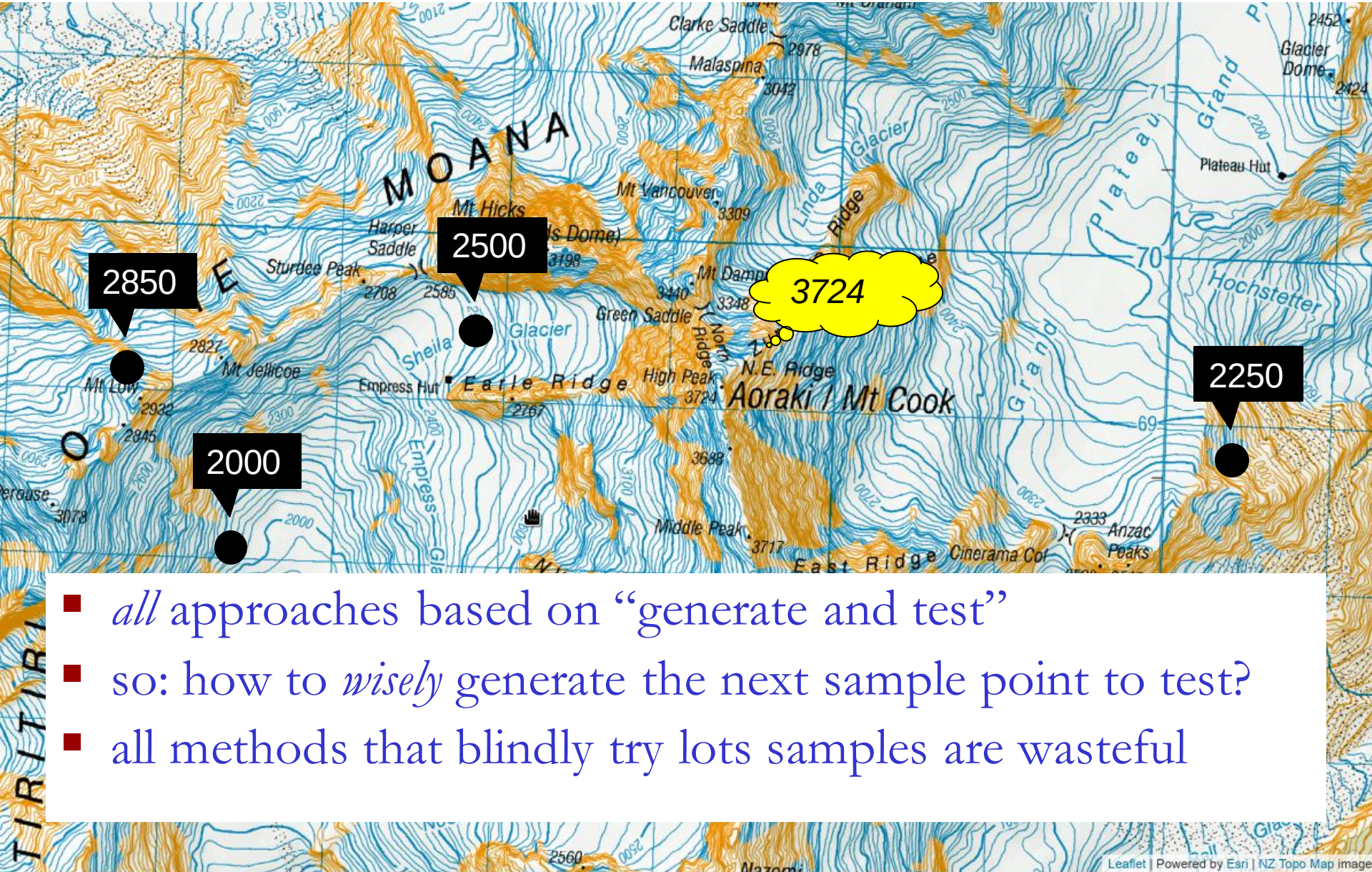
optimization : find the top of the hill



optimization : depends where you start



suppose *all you get* is elevation, at specific points



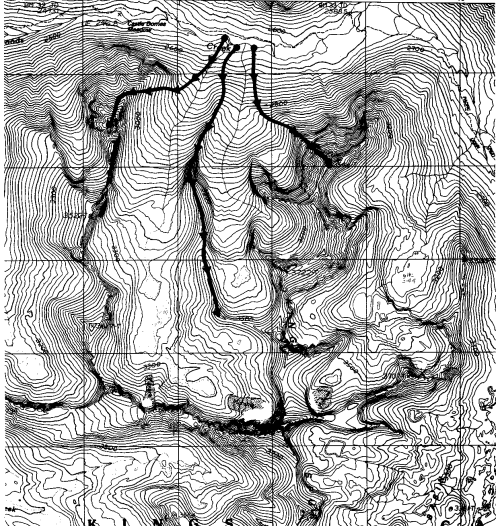
- *all* approaches based on "generate and test"
- so: how to *wisely* generate the next sample point to test?
- all methods that blindly try lots samples are wasteful

eg: optimizations that are "sample expensive"

application	search space	per-sample expense
new hull for boat, wing for plane…	shape parameters (e.g. 18 for hull)	• build & test it? • fancy simulation?
design a new robot	joint lengths, torque ratios, weight spread…	• just build & see? • build simulation, and test in that?
tuning any complex "engine" (*eg.* any AI)	various parameters	Testing takes time / money / effort.

- *all* approaches based on "generate and test"
- so: how to *wisely* generate the next sample point to test?
- all methods that blindly try lots samples are wasteful

Some heuristic optimization ideas



- Try to go up (or down) hills…
- Don't revisit previous points…?
- Explore around the best point so far…?
- Sometimes, try something *completely* new…(when?)
- Balance exploitation versus exploration (somehow?)...

→ hill-climbing, evolutionary algorithms, etc.....

these seem sensible, yet… *ad hoc*.

ideally:

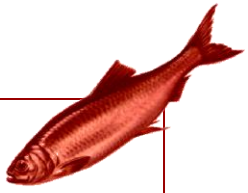
- those behaviours should be emergent, not "wired in" by us
- algorithm should learn surface properties as it goes

Optimizing an unknown function ("surface")

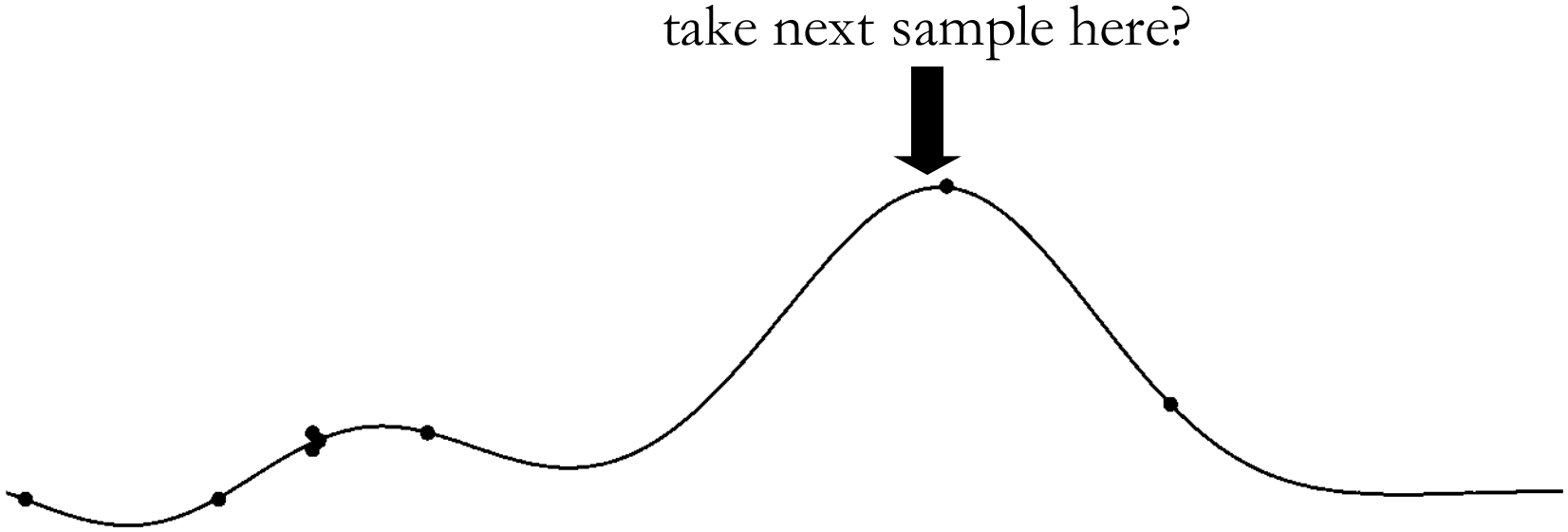
- find the x that maximizes y(x), just by measuring y at some test points x.
- assume as little as possible about the surface *a priori*.
- get there as quickly as possible, e.g.
  - min computation
  - min # samples

- min computation case is easy: take *lots* of samples
  - fine if data is "free"
  - what if it's not?
- min # samples:
  - `make the most of every piece of information about the surface`

Note: it doesn't matter where the latest point was

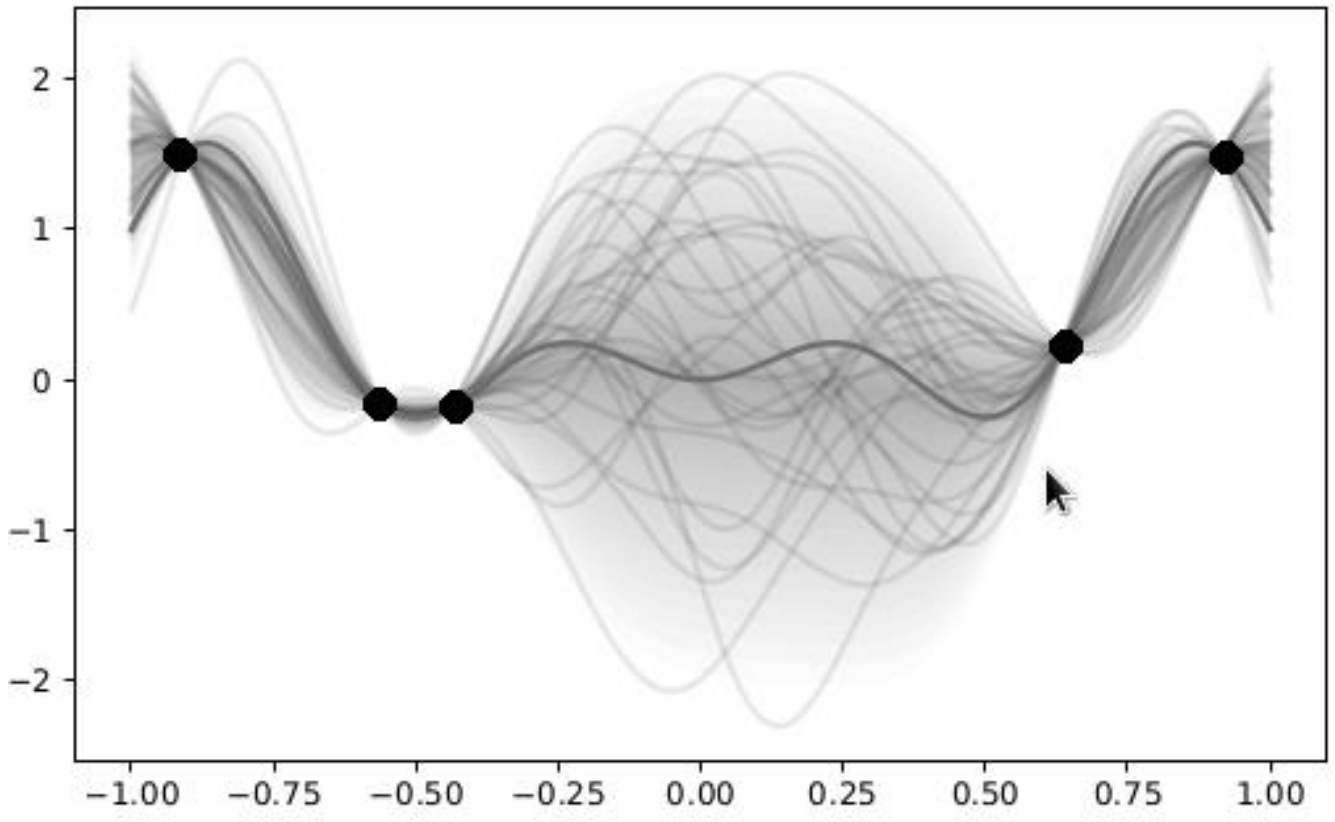


idea: fit existing data with a plausible "surrogate"



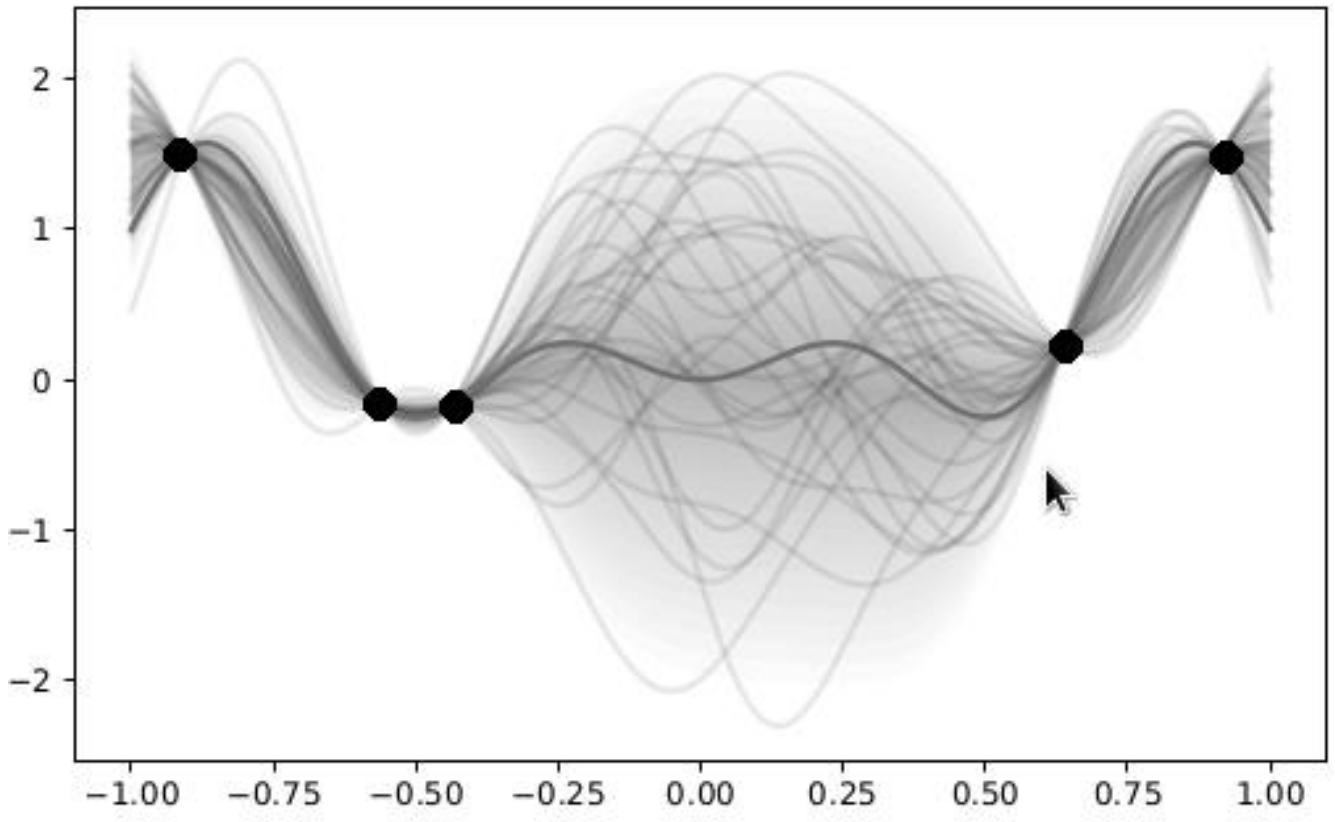
- there's a whole family of functions that could go through some data
- we could pick the "best" one, and try at its peak…
- bad idea

better idea: fit a *family* of plausible surrogates



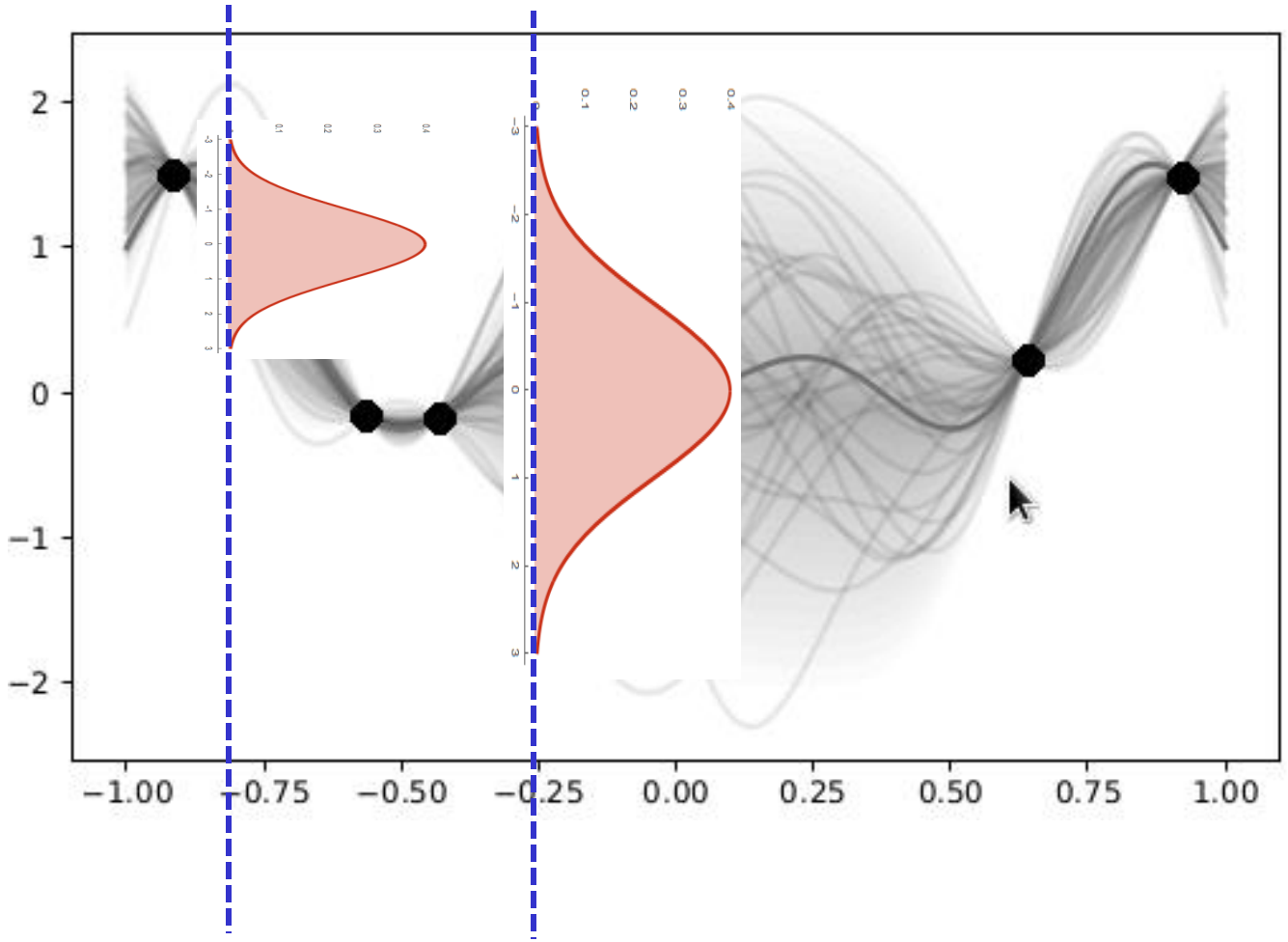
- there's a whole family of functions that could go through some data….
- new data points constrain the family more and more
- "error bars" rise as you move away from test points

how to do that (*not examinable*)

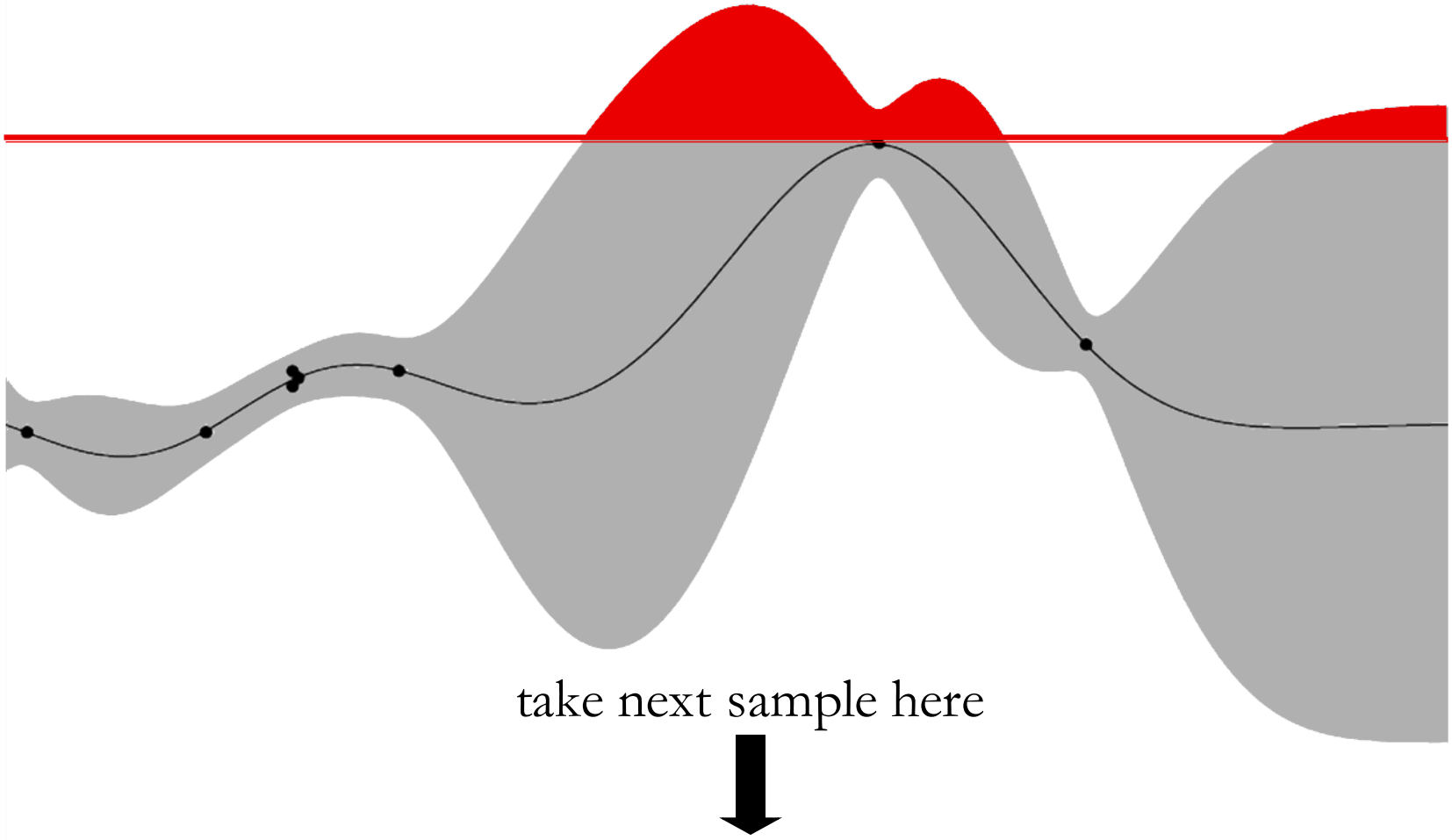


- there are several possibilities
- details are beyond us here, but for example we could:
  - ✓ e.g. apply a machine learning algorithm + randomness
  - ✓ e.g. apply "Bayesian" statistical methods instead

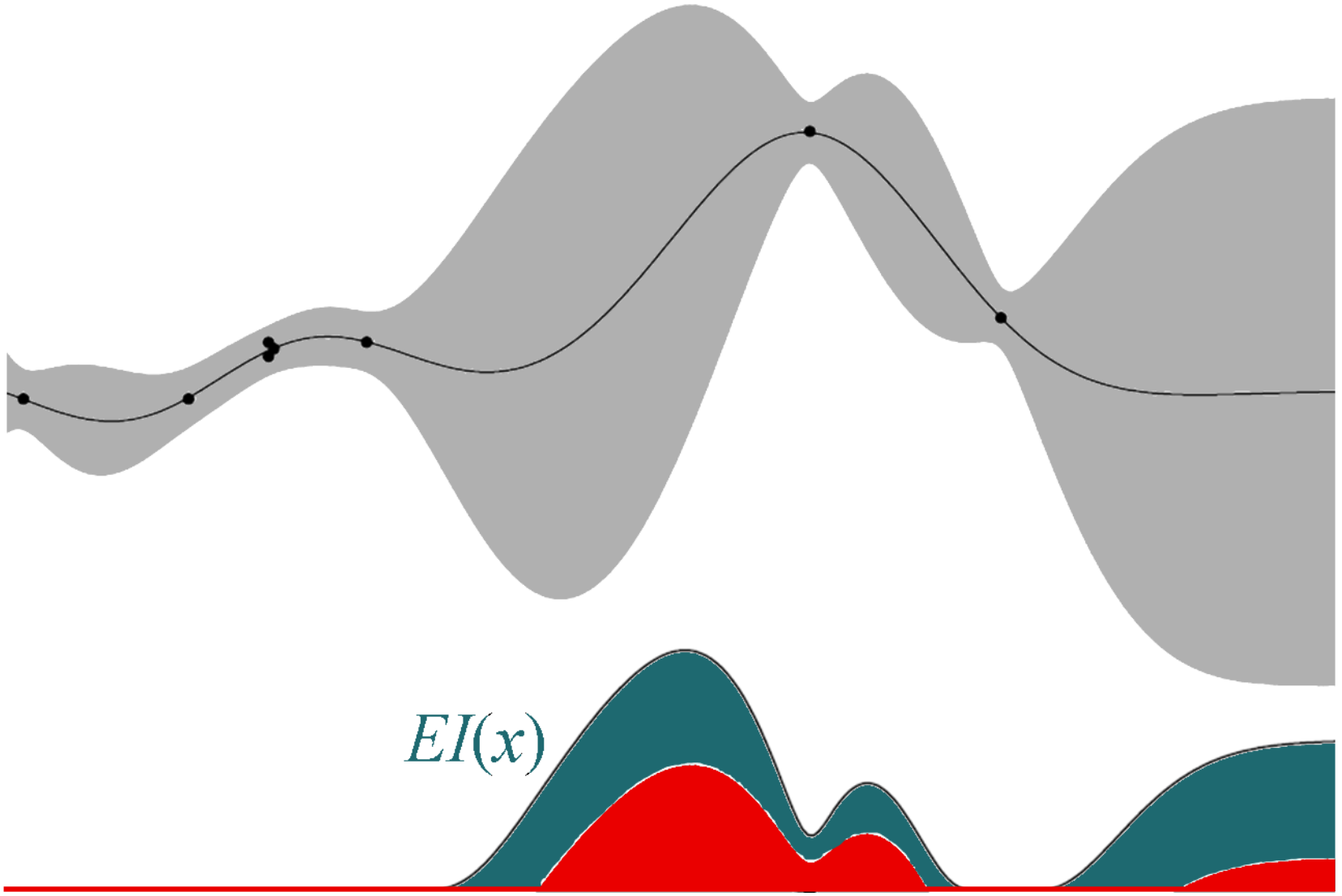
how to use the predicted distribution of values?



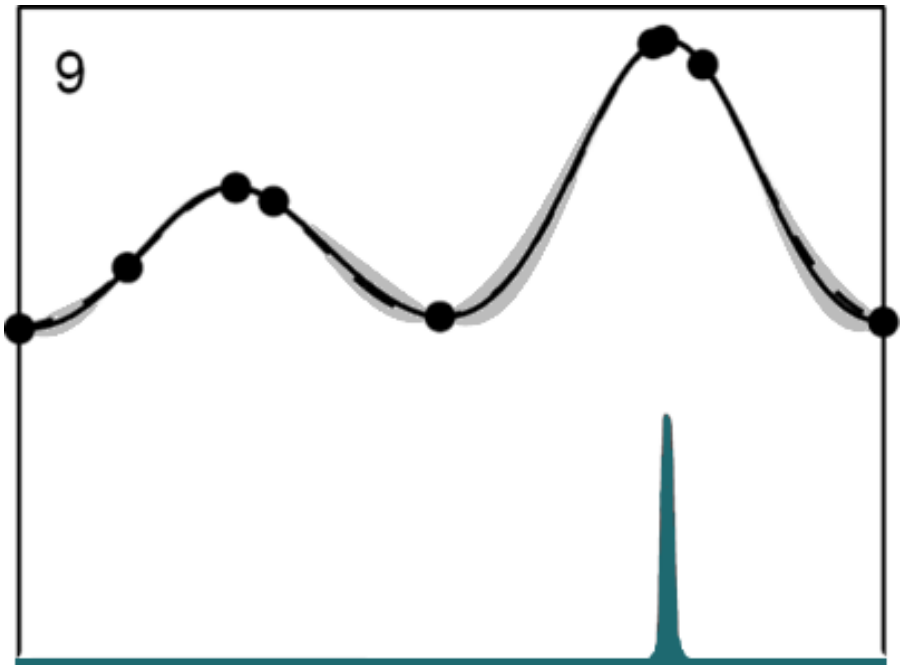
how to use the predicted distribution of values?



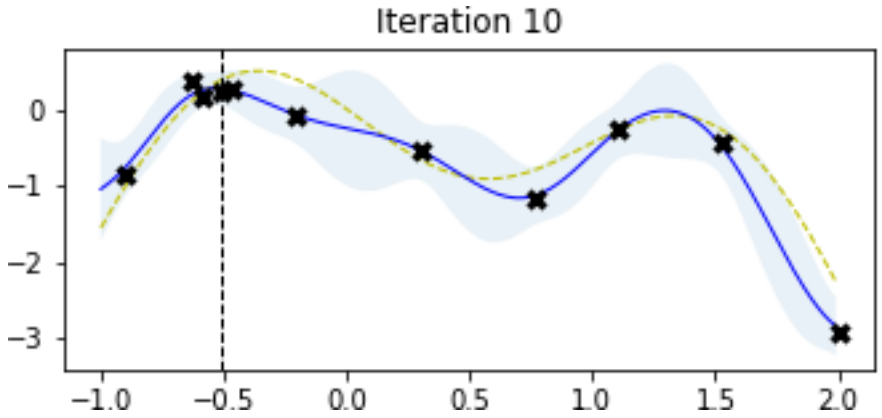
how to use the predicted distribution of values?

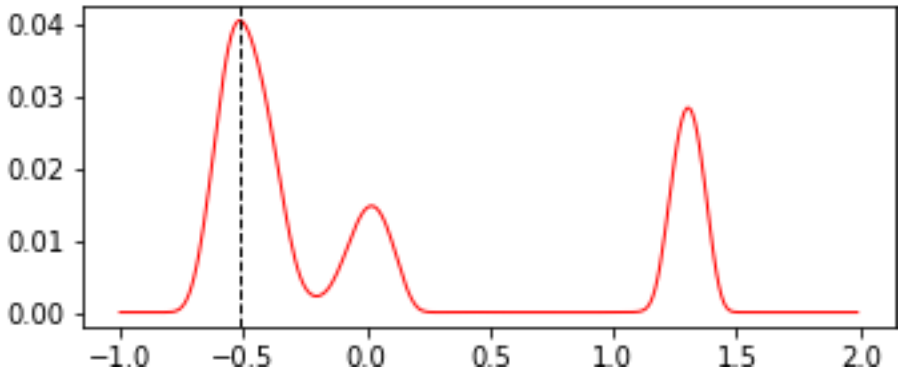


example in 1 dimension



another example in 1 dimension



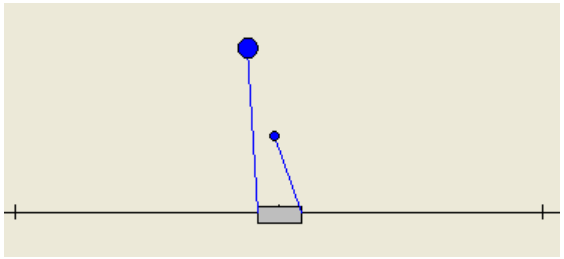


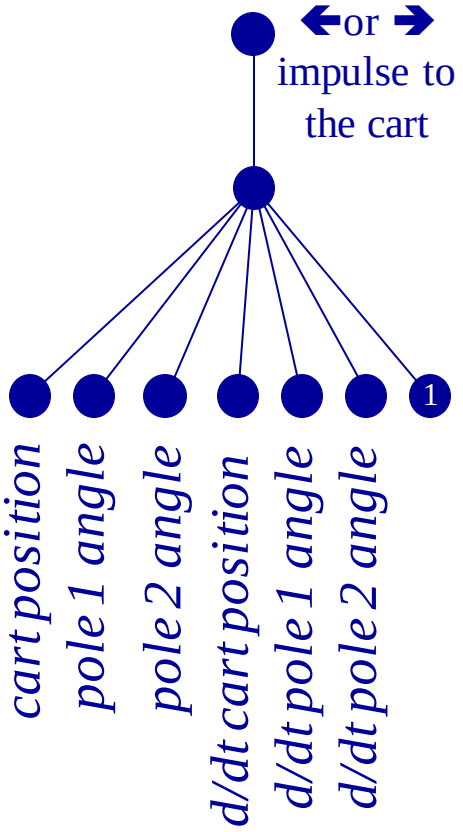
the "acquisition function" (eg. probability of improvement)

(made by running this python notebook)

example in 8d: learning to balance 2 poles (at once)

- two poles on one cart
- cart has to stay "in bounds"
- learner knows the cart position, the pole angles, and their rates of change
- evaluation: average time spent near-vertical, from a near-vertical start
- measuring "performance" of a robot is really expensive

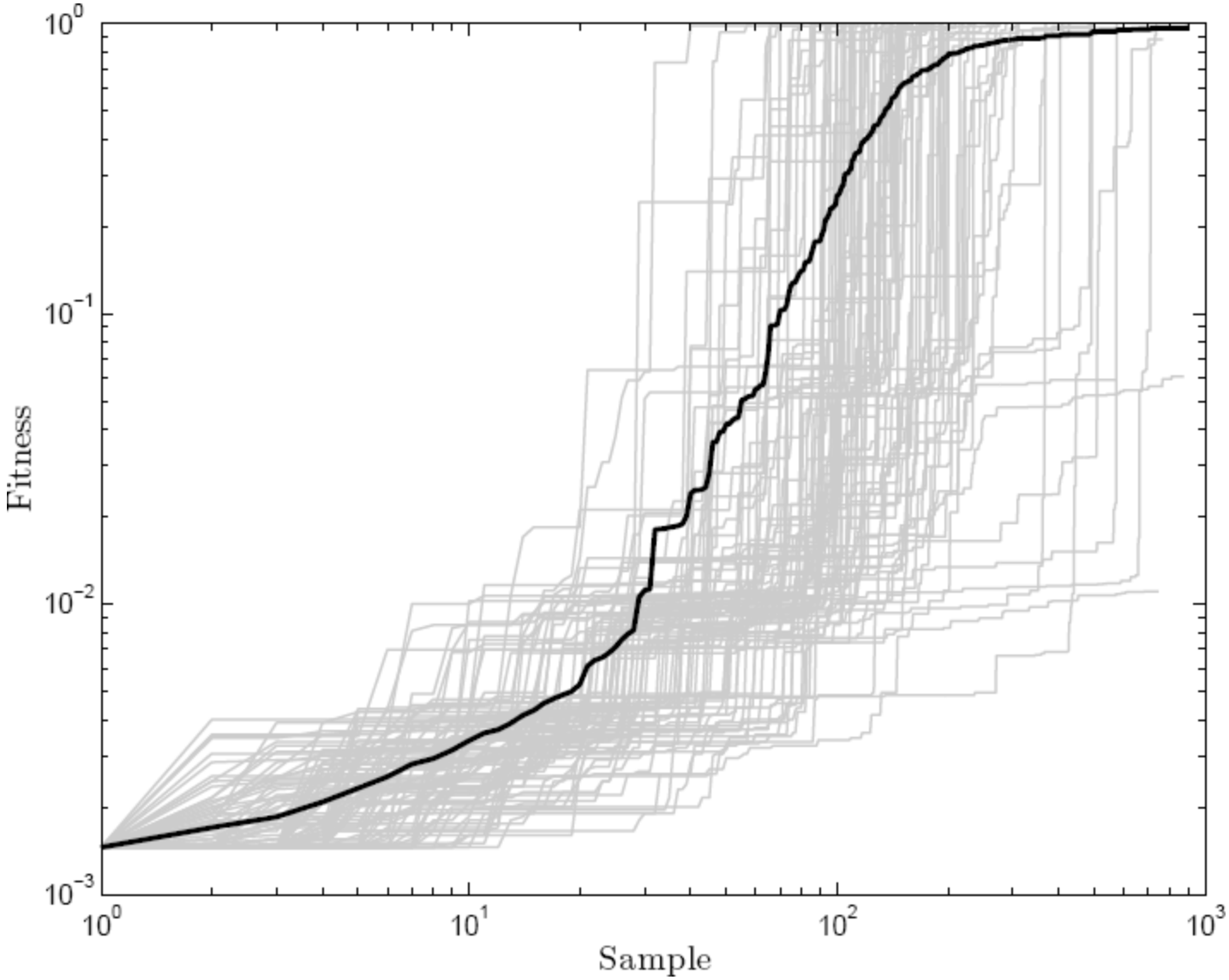




- controller is to be a neural network
- →a search problem in 8 dimensions
- the competition takes ~3500 samples to find a decent controller

example: learning to balance 2 poles at once

- we take ~150 samples
- cf. best opposition takes ~3500 samples



name of this algorithm: Bayesian Optimization

(the "Bayesian" refers to Bayes theorem, which is usually the basis for generating a *family* of surfaces, as needed)

- **Advantages**:
  - fewest samples to get to a good solution!
- **Drawbacks**?
  - need to make that family, somehow
  - need to optimize over the Prob-of-improvement surface, which could itself be a tough problem (although cheaper…)
  - both the above get harder in high dimensions ☹

- **`see also`**: if you like python notebooks, I highly recommend trying this one (from a blog by Martin Krasser). There's also this.