Lecture 8

Simulated Annealing

(and Information theory and Combinatoric Optimization)

Last Times:

- Regularization
- log-loss for logistic regression
- KL-Divergence
- p as empirical distribution
- Cross-entropy
- Entropy and MAXENT

Today

- likelihood ratio, deviance
- AIC and variable selection
- local search with random starts
- simulated annealing
- combinatoric search with simulated annealing

Model Comparison: Likelihood Ratio

$H(p)$ cancels out!!

$$D_{KL}(p,q) - D_{KL}(p,r) = H(p,q) - H(p,r) = E_p[log(r) - log(q)] = E_p[log(\frac{r}{q})]$$

In the sample approximation we have:

$$D_{KL}(p,q) - D_{KL}(p,r) = \frac{1}{N}\sum_i log(\frac{r_i}{q_i}) = \frac{1}{N} log(\frac{\prod_i r_i}{\prod_i q_i}) = \frac{1}{N} log(\frac{\mathcal{L}_r}{\mathcal{L}_q})$$

Model Comparison: Deviance

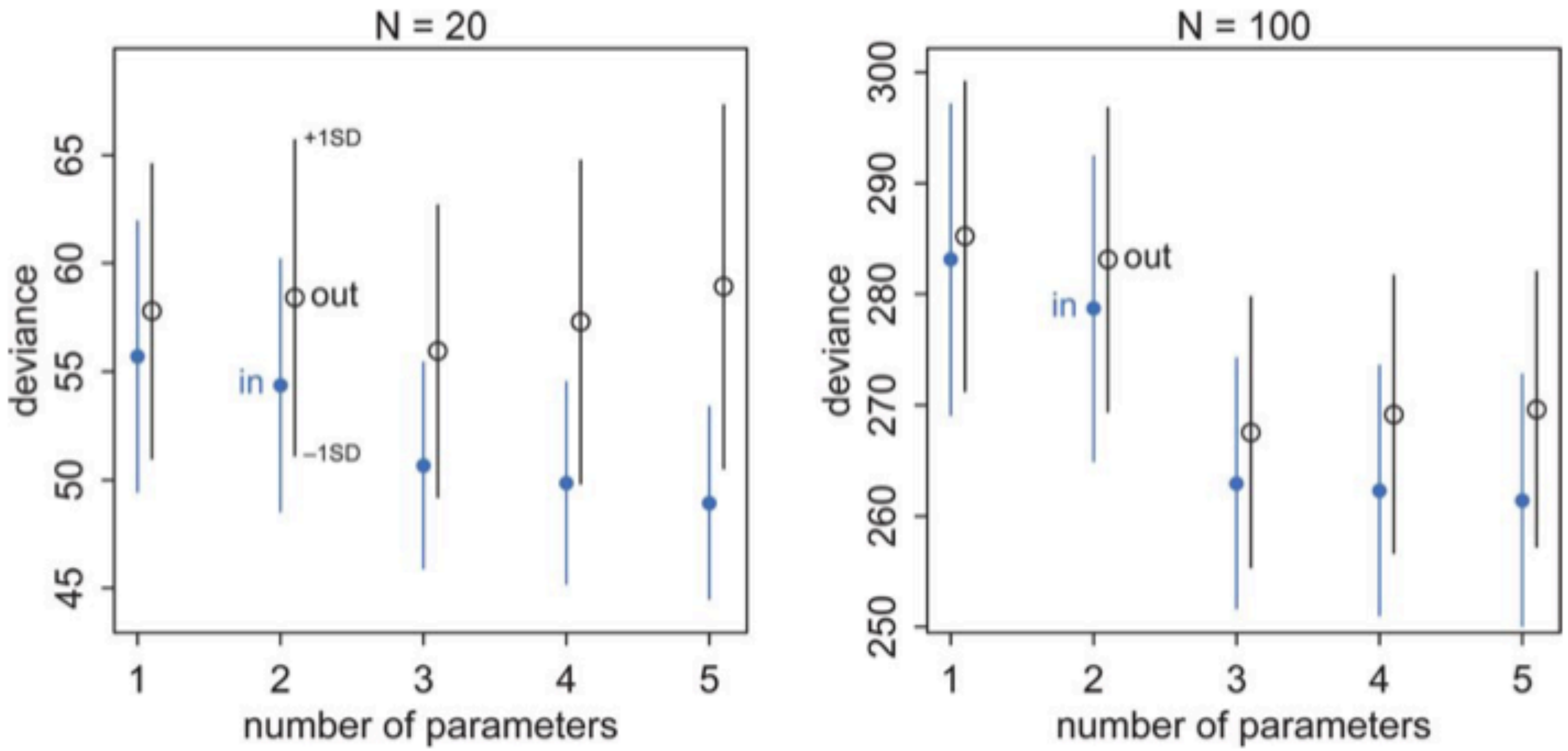
You only need the sample averages of the logarithm of r and q:

$$D_{KL}(p,q) - D_{KL}(p,r) = \langle log(r) \rangle - \langle log(q) \rangle$$

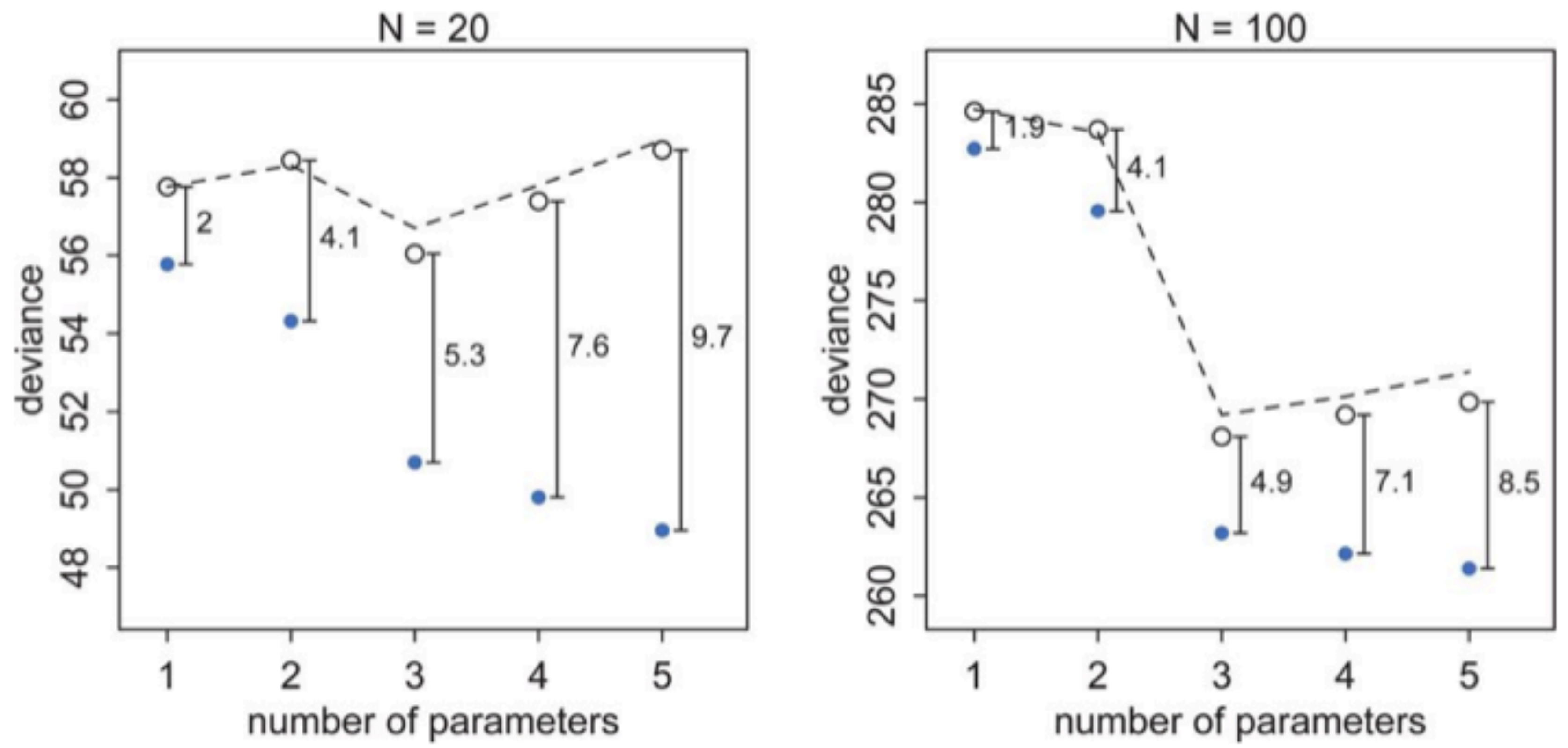
Define the deviance: $D(q) = -2\sum_i log(q_i)$, a risk (e.g., -2 $\times$ ℓ, although the distribution need not be a likelihood)...

$$D_{KL}(p,q) - D_{KL}(p,r) = \frac{2}{N}(D(q) - D(r))$$

Train to Test



AIC



The test set deviances are $2 * p$ above the training set ones.

Akake **Information Criterion**:

$$AIC = D_{train} + 2p$$

- Assumption: likelihood is approximately multivariate gaussian.
- high p increases the out-of-sample deviance, less desirable.

AIC **estimates out-of-sample deviance**

REGULARIZATION

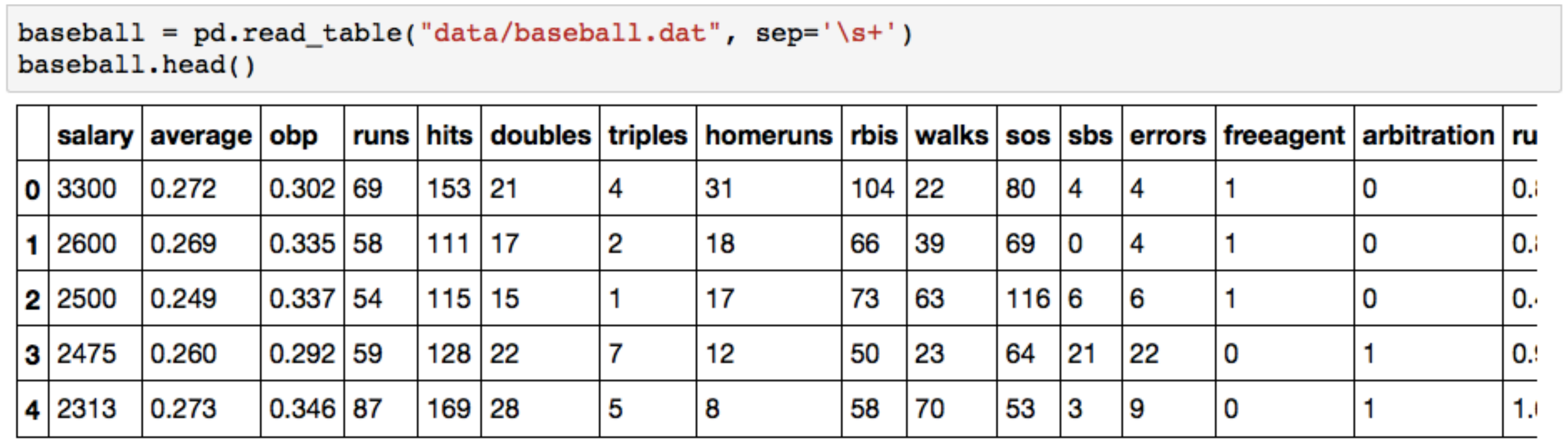
$$AIC = -2\ell + 2p$$

$$\mathcal{R}(h_j) = \sum_{y_i \in \mathcal{D}} (y_i - h_j(x_i))^2 + p$$

- penalized log-likelihood or risk if we choose to identify our distribution with the likelihood: REGULARIZATION

Baseball data set

Description: Salaries in 1992 and 27 performance statistics for 337 baseball players (no pitchers) in 1991.



```
baseball = pd.read_table("data/baseball.dat", sep='\s+')
baseball.head()
```

	salary	average	obp	runs	hits	doubles	triples	homeruns	rbis	walks	sos	sbs	errors	freeagent	arbitration	ru
0	3300	0.272	0.302	69	153	21	4	31	104	22	80	4	4	1	0	0.
1	2600	0.269	0.335	58	111	17	2	18	66	39	69	0	4	1	0	0.
2	2500	0.249	0.337	54	115	15	1	17	73	63	116	6	6	1	0	0.
3	2475	0.260	0.292	59	128	22	7	12	50	23	64	21	22	0	1	0.
4	2313	0.273	0.346	87	169	28	5	8	58	70	53	3	9	0	1	1.

(from http://www.amstat.org/publications/jse/v6n2/datasets.watnik.html)

AIC for Linear Regression

$$AIC = D_{train} + 2p \text{ where } D(q) = -2\sum_i log(q_i) = -2\ell$$

$$\sigma^2_{MLE} = \frac{1}{N} SSE$$

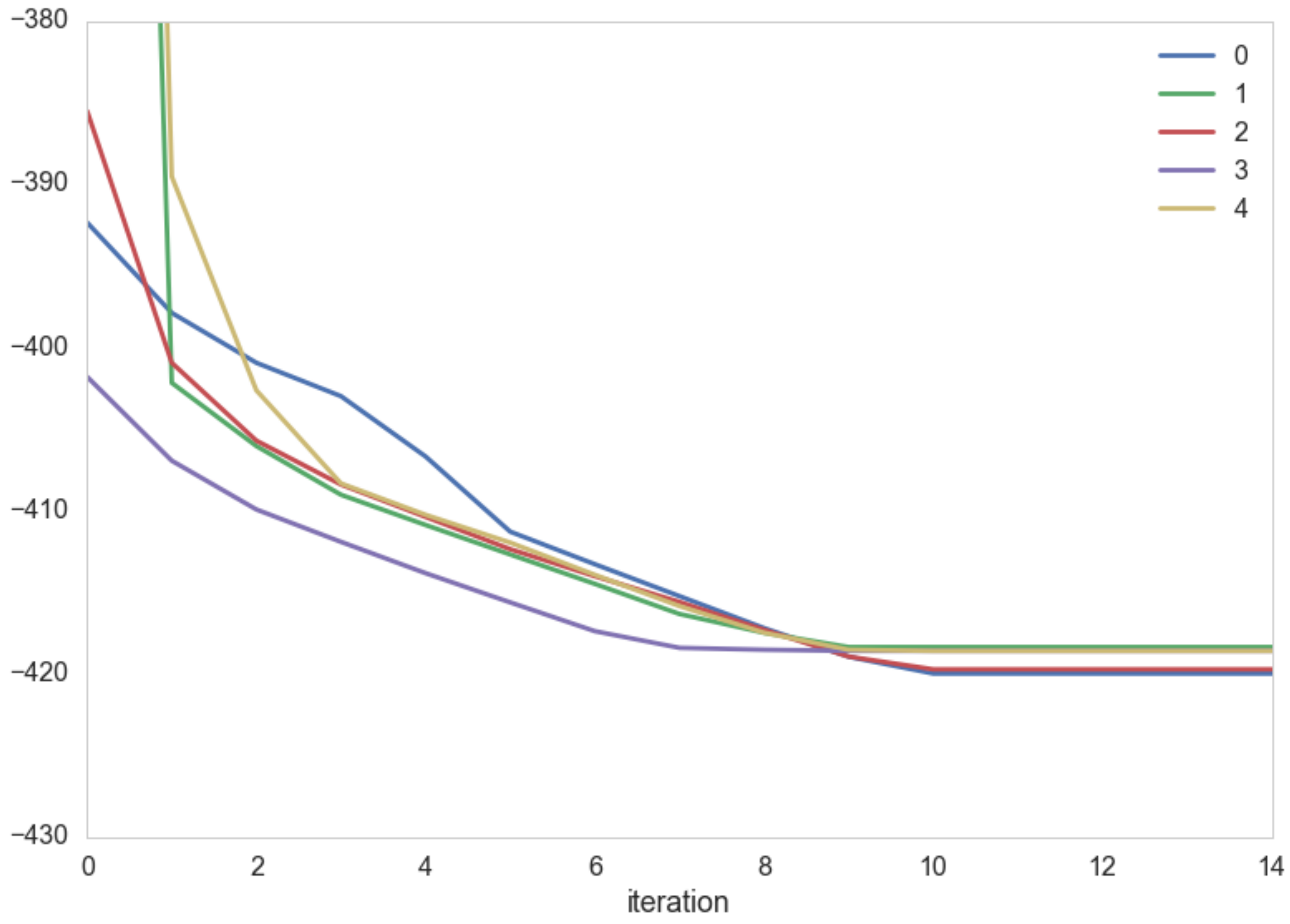
$$AIC = -2(-\frac{N}{2}(log(2\pi) + log(\sigma^2)) - 2(-\frac{1}{2\sigma^2_{MLE}} \times SSE) + 2p$$

$$AIC = Nlog(SSE/N) + 2p + constant$$

Local Search with Random starts

- wish to find best set of features for prediction
- want parsimonius model, no overfitting
- Combinatoric search is hard
- 2^{27} sized search space for baseball problem
- make local perturbations

```python
from sklearn.linear_model import LinearRegression

runs_aic = np.empty((nstarts, iterations))

for i in range(nstarts):

    run_current = runs[i]

    for j in range(iterations):

        # Extract current set of predictors
        run_vars = predictors[predictors.columns[run_current]]
        g = LinearRegression().fit(X=run_vars, y=logsalary)
        run_aic = aic(g, run_vars, logsalary)
        run_next = run_current

        # Test all models in 1-neighborhood and select lowest AIC
        for k in range(ncols):
            run_step = run_current.copy()
            run_step[k] = not run_current[k]
            run_vars = predictors[predictors.columns[run_step]]
            g = LinearRegression().fit(X=run_vars, y=logsalary)
            step_aic = aic(g, run_vars, logsalary)
            if step_aic < run_aic:
                run_next = run_step.copy()
                run_aic = step_aic

        run_current = run_next.copy()
        runs_aic[i,j] = run_aic

    runs[i] = run_current
```



```
FEATURES

arbitration      5   BEST SOLUTION: (array([ 1,  2,  5,  7,  9, 11, 12, 13, 14, 15, 24, 25]),)
rbis             5
freeagent        5
obppererror      4       0 -420.000042669
sos              4       1 -418.380197435
hitsperso        3       2 -419.743167044
sbshits          3       3 -418.611888647
hitspererror     3       4 -418.611888647
sbsobp           3       AICs
soserrors        2
runs             2
hrsperso         2
```

Features present in most starts, left, best solution, right top, AICs, right

Physical Annealing

A system is first heated to a melting state and then cooled down slowly.

- when solid is heated, its molecules start moving randomly, and its energy increases
- if subsequent process of cooling is slow, the energy decreases slowly, with some random increases governed by the Boltzmann distribution
- if cooling slow and deep enough, system will eventually settle down to the lowest energy state with minimal potential energy

Simulated Annealing

Minimize f by identifying with the energy of an imaginary physical system undergoing an annealing process.

Move from x_i to x_j via a **proposal**.

If the new state has lower energy, accept x_j.

If the new state has higher energy, accept with probability

$$A = \exp\left(-\Delta f/kT\right)$$

- stochastic acceptance of higher energy states, allows our process to escape local minima.
- When T is high, the acceptance of these uphill moves is higher, and local minima are discouraged.
- As T is lowered, more concentrated search near current local minimum, since only few uphill moves will be allowed.
- Thus, if we get our temperature decrease schedule right, we can hope that we will converge to a global minimum.

If the lowering of the temperature is sufficiently slow, the system reaches "thermal equilibrium" at each temperature. Then Boltzmann's distribution applies:

$$p(X = i) = \frac{1}{Z(T)} \exp\left(\frac{-E_i}{kT}\right)$$

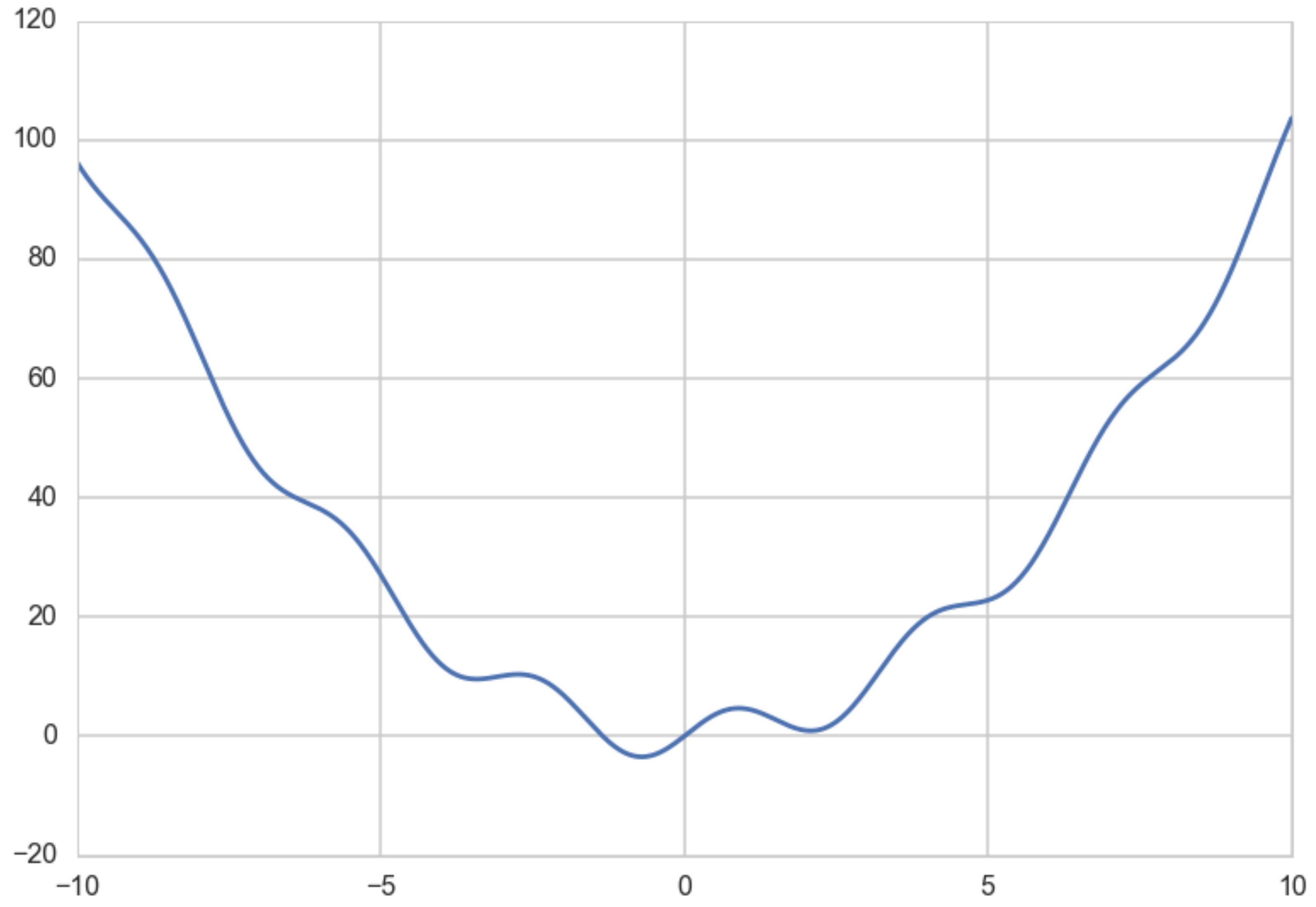
where

$$Z(T) = \sum_j \exp\left(\frac{-E_j}{kT}\right)$$

Proposal

- it proposes a new position x from a **neighborhood** $\mathcal{N}$ at which to evaluate the function.
- all the positions x in the domain we wish to minimize a function f over ought to be able to communicate.
- detailed balance: proposal is symmetric
- ensures $\{x_t\}$ generated by simulated annealing is a stationary markov chain with target boltzmann distribution: equilibrium

Example: $x^2 + 4sin(2x)$



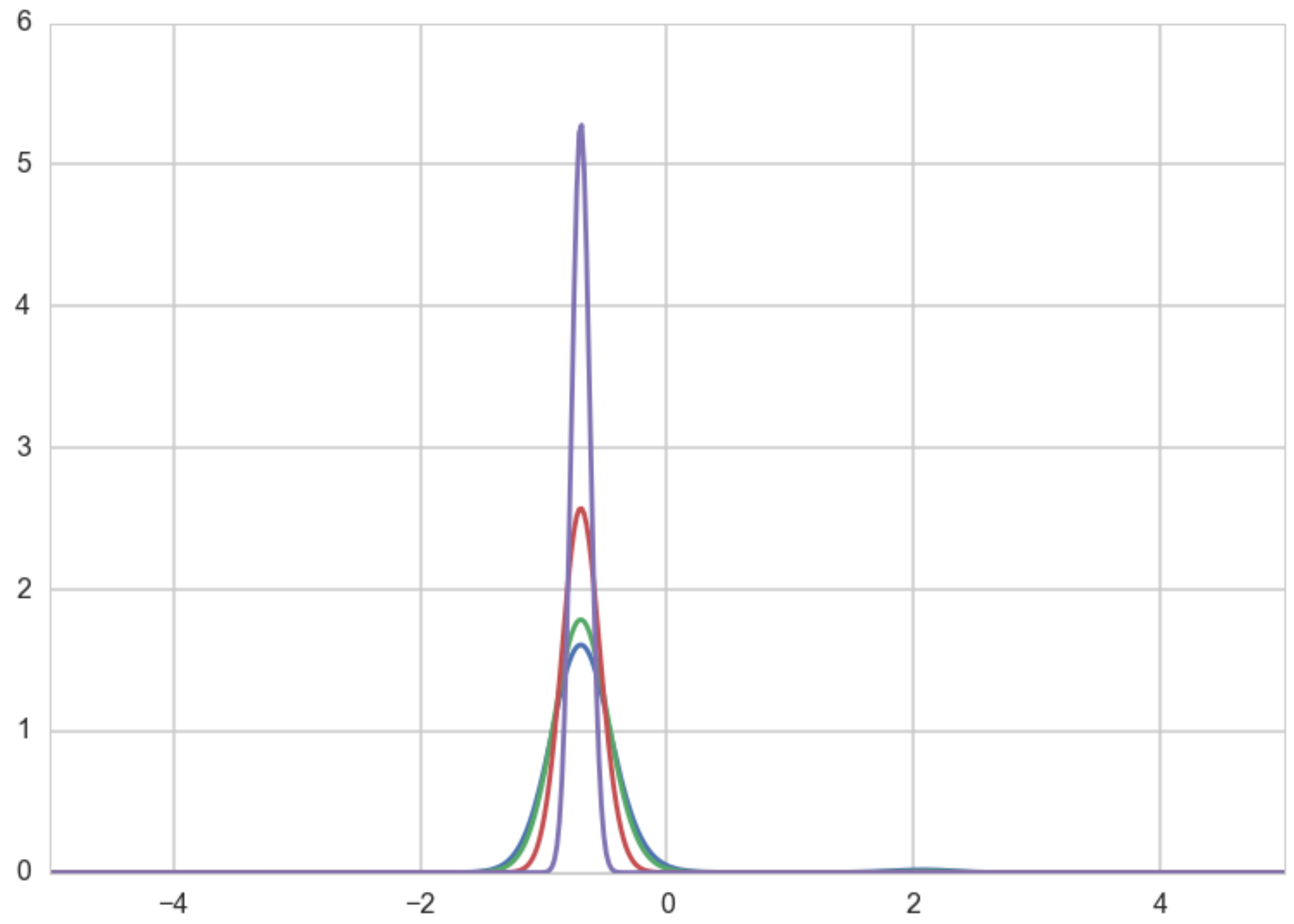
If you identify

$$p_T(x) = e^{-f(x)/T} \text{ and } p(x) = e^{-f(x)}$$

Then:

$$P_T(x) = P(x)^{1/T}$$

- you get a peakier distribution as $T to 0$ around the global minimum: distribution $\rightarrow$ optimum!
- the globality and the exponentiation ensures that this peak is favored over the rest in f



Normalized Boltzmann distribution

- M global minima in set $\mathcal{M}$
- function minimum value f_{min}:

$$p(x_i) = \frac{e^{-(f(x_i)-f_{min})/T}}{M + \sum_{j \notin \mathcal{M}} e^{-(f(x_i)-f_{min})/T}}$$

As $T \to 0$ from above, this becomes $1/M$ if $x_i \in \mathcal{M}$ and 0 otherwise.

The Simulated Annealing Algorithm

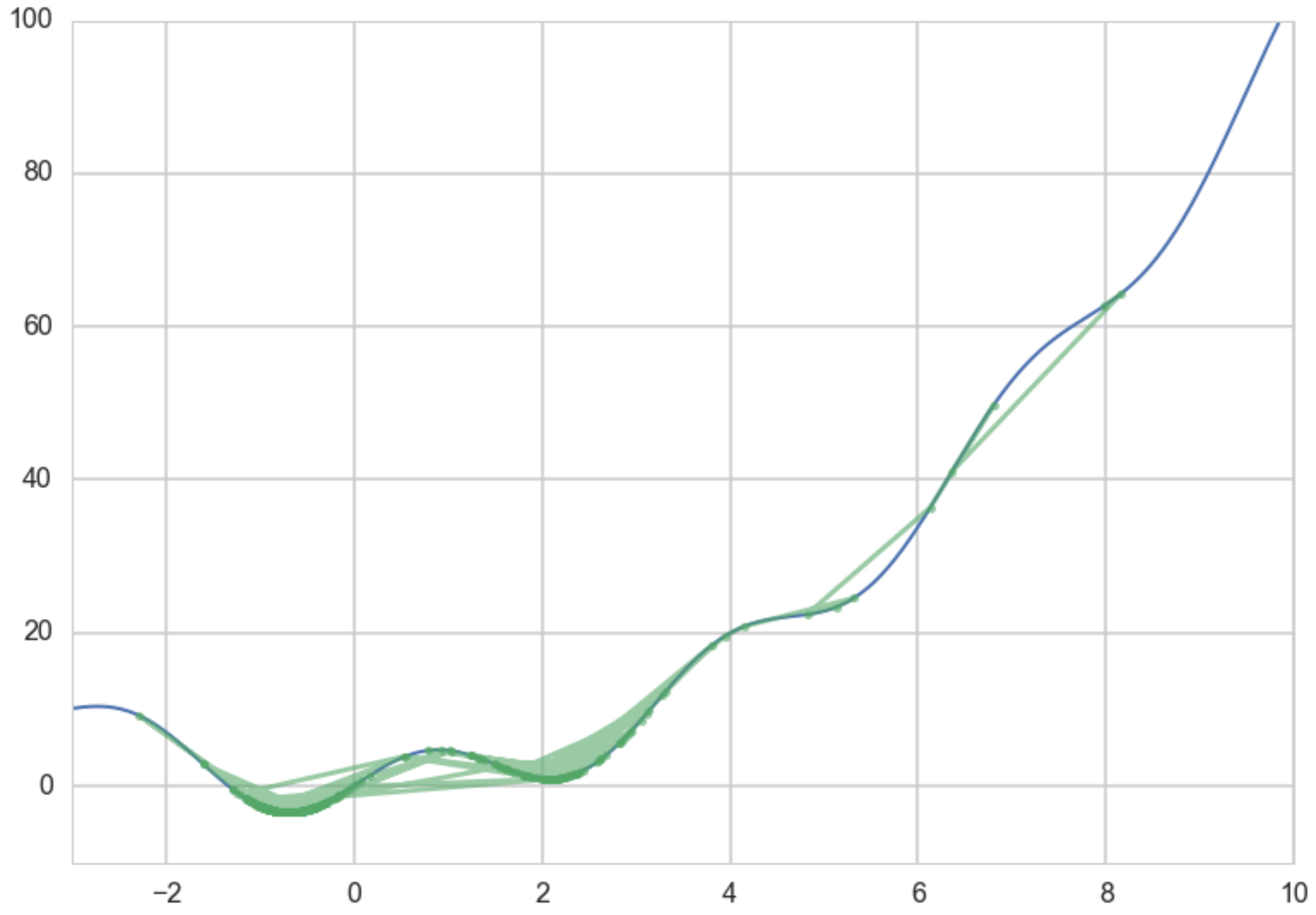
1. Initialize x_i, $T, L(T)$ where L = iterations at a particular temperature.

2. Perform L transitions:

(a) propose x_j (b) If x_j is accepted (according to probability $P = e^{(-\Delta E/T)}$, set $x_{i+1} = x_j$, else set $x_{i+1} = x_i$

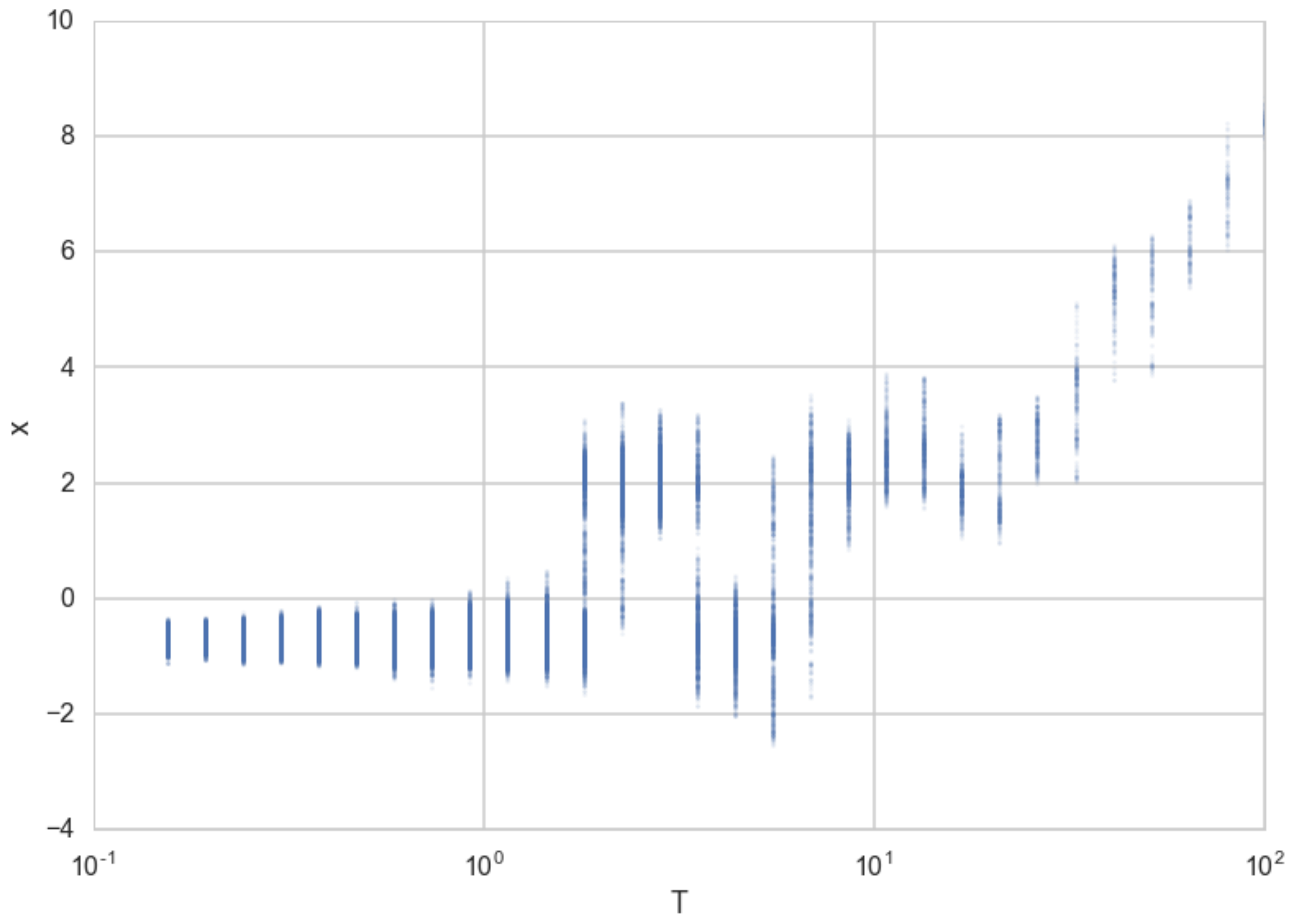
3. Update T and L, go to 2

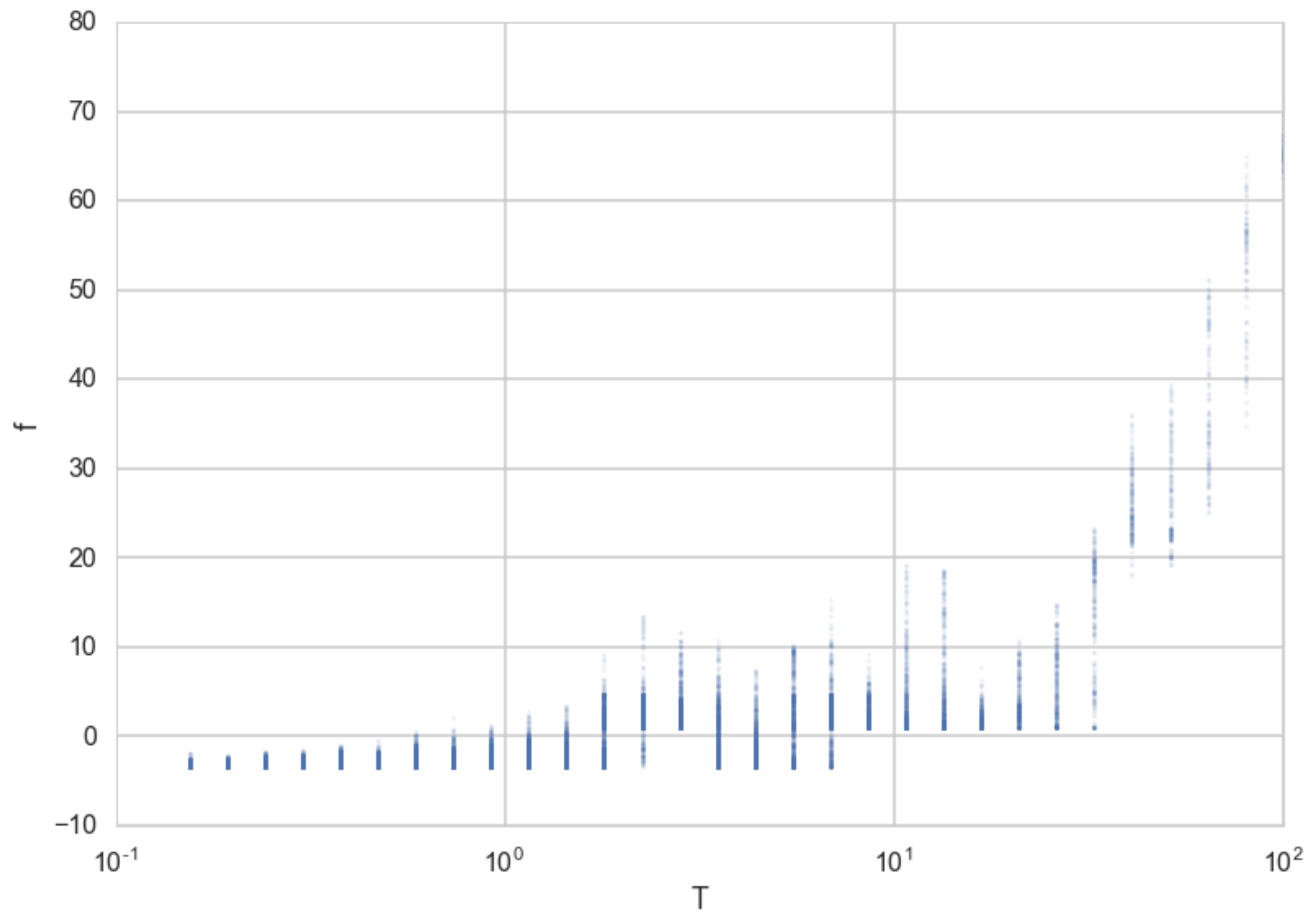
```python
def sa(energyfunc, initials, epochs, tempfunc, iterfunc, proposalfunc):
    accumulator=[]
    best_solution = old_solution = initials['solution']
    T=initials['T']
    length=initials['length']
    best_energy = old_energy = energyfunc(old_solution)
    accepted=0
    total=0
    for index in range(epochs):
        print("Epoch", index)
        if index > 0:
            T = tempfunc(T)
            length=iterfunc(length)
        print("Temperature", T, "Length", length)
        for it in range(length):
            total+=1
            new_solution = proposalfunc(old_solution)
            new_energy = energyfunc(new_solution)
            # Use a min here as you could get a "probability" > 1
            alpha = min(1, np.exp((old_energy - new_energy)/T))
            if ((new_energy < old_energy) or (np.random.uniform() < alpha)):
                # Accept proposed solution
                accepted+=1
                accumulator.append((T, new_solution, new_energy))
                if new_energy < best_energy:
                    # Replace previous best with this one
                    best_energy = new_energy
                    best_solution = new_solution
                    best_index=total
                    best_temp=T
                old_energy = new_energy
                old_solution = new_solution
            else:
                # Keep the old stuff
                accumulator.append((T, old_solution, old_energy))

    best_meta=dict(index=best_index, temp=best_temp)
    print("frac accepted", accepted/total, "total iterations", total, 'bmeta', best_meta)
    return best_meta, best_solution, best_energy, accumulator
```

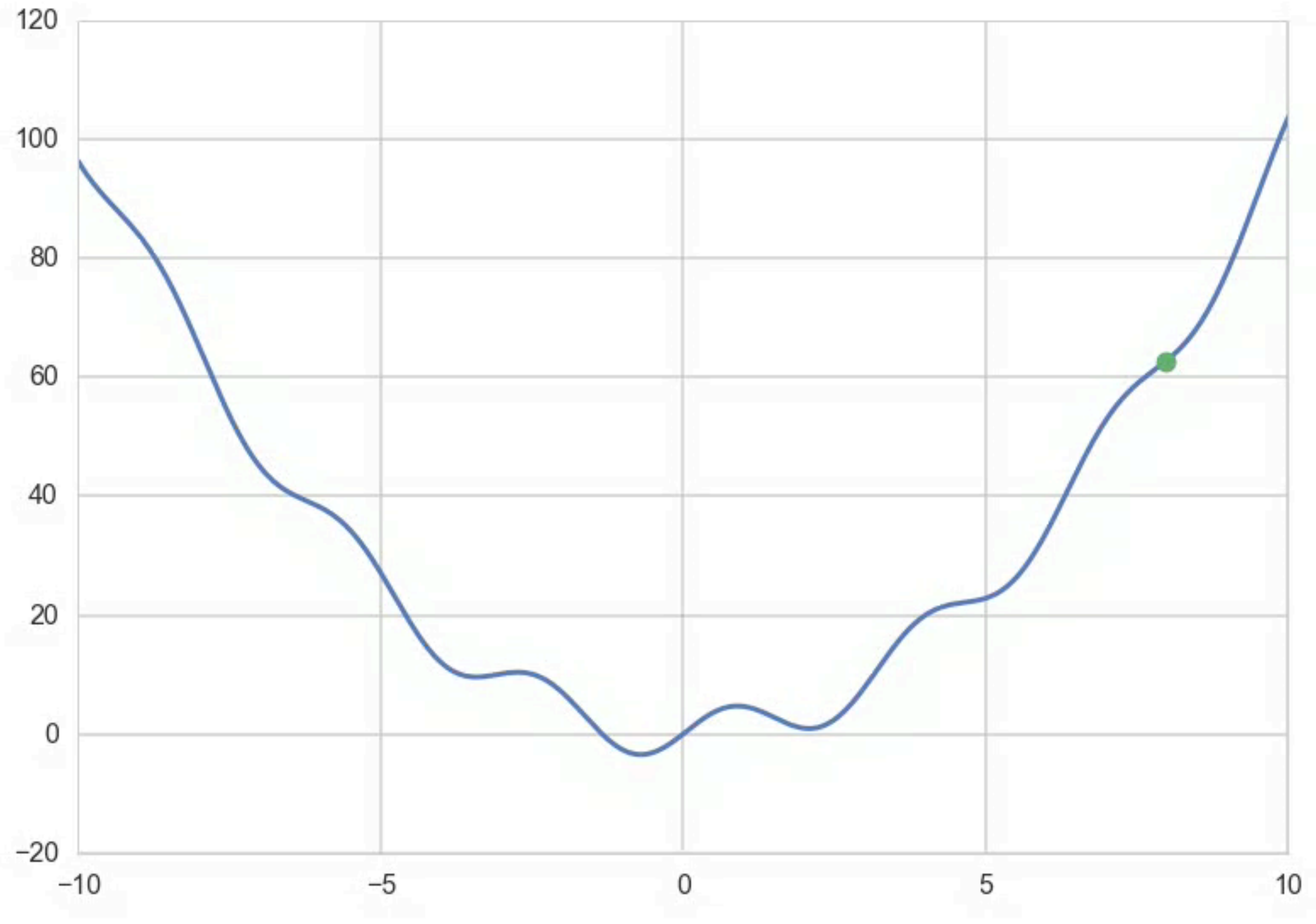


```python
tf = lambda t: 0.8*t #temperature function
itf = lambda length: math.ceil(1.2*length) #iteration function
inits=dict(solution=8, length=100, T=100)
bmeta, bs, be, out = sa(f, inits, 30, tf, itf, pf)
```

```
Epoch 0
Temperature 100 Length 100
Epoch 1
Temperature 80.0 Length 120
Epoch 2
Temperature 64.0 Length 144
Epoch 3
Temperature 51.2 Length 173
Epoch 4
Temperature 40.96000000000001 Length 208
Epoch 5
Temperature 32.76800000000001 Length 250
Epoch 6
Temperature 26.21440000000001 Length 300
Epoch 7
Temperature 20.97152000000001 Length 360
...
Epoch 27
Temperature 0.24178516392292618 Length 13863
Epoch 28
Temperature 0.19342813113834095 Length 16636
Epoch 29
Temperature 0.15474250491067276 Length 19964
frac accepted 0.7921531132581857 total iterations 119232 bmeta {'index': 112695, 'temp': 0.15474250491067276}
```


10
8
6
4
2
0
−2
−4
x
10^{-1}
10^{0}
10^{1}
10^{2}
T


80
70
60
50
40
30
20
10
0
−10
f
10^{-1}
10^{0}
10^{1}
10^{2}
T


120
100
80
60
40
20
0
−20
−10
−5
0
5
10

Practical choices

- Start T_0 large to accept all transitions.
- Thermostat

1. Linear: Temperature decreases as $T_{k+1} = \alpha T_k$.
2. Exponential: Temperature decreases as 0.95^{k}
3. Logarithmic: Temperature decreases as $1/\log(\mathrm{k})$

- Reannealing interval, or epoch length is the number of points to accept before reannealing (change the temperature). Typical starting value is 100, increase it as $L_{k+1} = \beta L_k$ where $\beta > 1$.
- Larger decreases in temperature require correspondingly longer epoch lengths to re-equilibriate
- Running long epochs at larger temperatures is not very useful. Decrease temperature rapidly at first.

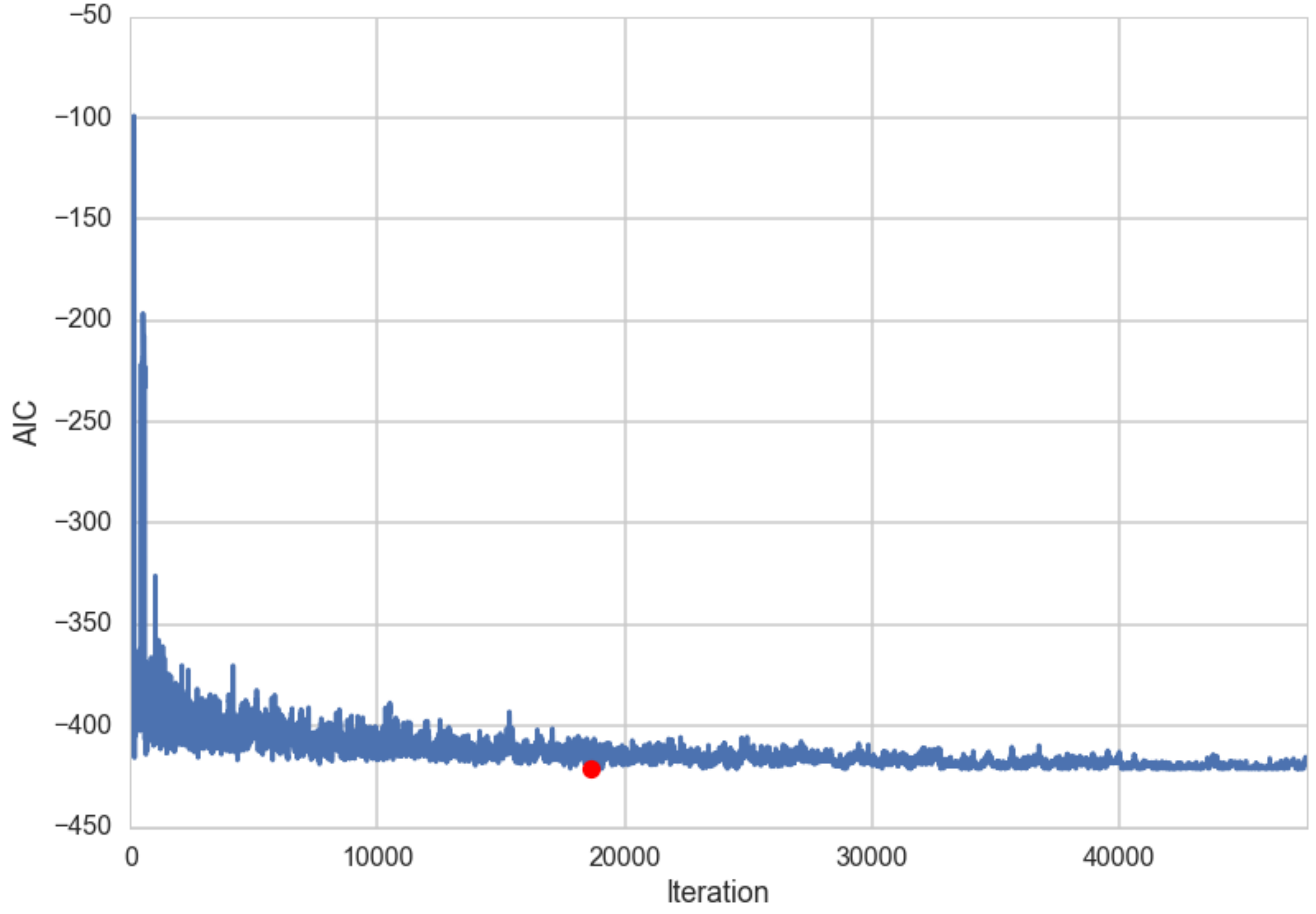
Simulated Annealing for baseball

```
bbinits=dict(solution=np.random.binomial(1, 0.5, ncols).astype(bool),
               length=100, T=100)
def efunc(solution):
   solution_vars = predictors[predictors.columns[solution]]
   g = LinearRegression().fit(X=solution_vars, y=logsalary)
   return aic(g, solution_vars, logsalary)
def pfunc(solution):
   flip = np.random.randint(0, ncols)
   solution_new = solution.copy()
   solution_new[flip] = not solution_new[flip]
   return solution_new
```

```
tf2 = lambda temp: 0.8*temp
itf2 = lambda length: math.ceil(1.2*length)
bb_bmeta, bb_bs, bb_be, bb_out = sa(efunc, bbinits, 25, tf2, itf2, pfunc)
```

```
Epoch 0
Temperature 100 Length 100
Epoch 1
Temperature 80.0 Length 120
Epoch 2
Temperature 64.0 Length 144
Epoch 3
Temperature 51.2 Length 173
Epoch 4
Temperature 40.96000000000001 Length 208
Epoch 5
Temperature 32.76800000000001 Length 250
...
Epoch 21
Temperature 0.9223372036854786 Length 4641
Epoch 22
Temperature 0.7378697629483829 Length 5570
Epoch 23
Temperature 0.5902958103587064 Length 6684
Epoch 24
Temperature 0.4722366482869651 Length 8021
frac accepted 0.37204513458427013 total iterations 47591 bmeta {'index': 18619, 'temp': 1.4411518807585602}
```

```
Best AIC: -420.9472114371548
Best solution: (array([ 1,  2,  5,  7,  9, 12, 13, 14, 15, 23, 24, 25]),)
Discovered at iteration 18618
```


AIC
Iteration