EE-559 – Deep learning

2. Learning, capacity, basic embeddings

François Fleuret

https://fleuret.org/dlc/

[version of: June 14, 2018]

idiap RESEARCH INSTITUTE

ÉCOLE POLYTECHNIQUE
FÉDÉRALE DE LAUSANNE

Learning from data

The general objective of machine learning is to capture regularity in data to make predictions.

In our regression example, we modeled age and blood pressure as being linearly related, to predict the latter from the former.

The general objective of machine learning is to capture regularity in data to make predictions.

In our regression example, we modeled age and blood pressure as being linearly related, to predict the latter from the former.

There are multiple types of inference that we can roughly split into three categories:

- Classification (*e.g.* object recognition, cancer detection, speech processing),
- regression (*e.g.* customer satisfaction, stock prediction, epidemiology), and
- density estimation (*e.g.* outlier detection, data visualization, sampling/synthesis).

The standard formalization considers a measure of probability

$$\mu_{X,Y}$$

over the observation/value of interest, and i.i.d. training samples

$$(x_n, y_n), \ n = 1, \ldots, N.$$

Intuitively, for classification it can often be interpreted as

$$\mu_{X,Y}(x, y) = \mu_{X|Y=y}(x)\, P(Y = y)$$

that is, draw Y first, and given its value, generate X.

Intuitively, for classification it can often be interpreted as

$$\mu_{X,Y}(x, y) = \mu_{X|Y=y}(x)\, P(Y = y)$$

that is, draw Y first, and given its value, generate X.

Here

$$\mu_{X|Y=y}$$

stands for the population of the observable signal for class y (*e.g.* "sound of an /ē/", "image of a cat").

For regression, one would interpret the joint law more naturally as

$$\mu_{X,Y}(x, y) = \mu_{Y|X=x}(y)\,\mu_X(x)$$

which would be: first, generate X, and given its value, generate Y.

For regression, one would interpret the joint law more naturally as

$$\mu_{X,Y}(x, y) = \mu_{Y|X=x}(y)\,\mu_X(x)$$

which would be: first, generate X, and given its value, generate Y.

In the simple cases

$$Y = f(X) + \epsilon$$

where f is the deterministic dependency between x and y, and ϵ is a random noise, independent of X.

With such a model, we can more precisely define the three types of inferences we introduced before:

Classification,

- (X, Y) random variables on $\mathcal{Z} = \mathbb{R}^D \times \{1, \ldots, C\}$,
- we want to estimate $\text{argmax}_y P(Y = y \mid X = x)$.

With such a model, we can more precisely define the three types of inferences we introduced before:

Classification,

- (X, Y) random variables on $\mathcal{Z} = \mathbb{R}^D \times \{1, \ldots, C\}$,
- we want to estimate $\text{argmax}_y P(Y = y \mid X = x)$.

Regression,

- (X, Y) random variables on $\mathcal{Z} = \mathbb{R}^D \times \mathbb{R}$,
- we want to estimate $\mathbb{E}(Y \mid X = x)$.

With such a model, we can more precisely define the three types of inferences we introduced before:

Classification,

- (X, Y) random variables on $\mathcal{Z} = \mathbb{R}^D \times \{1, \ldots, C\}$,
- we want to estimate $\text{argmax}_y P(Y = y \mid X = x)$.

Regression,

- (X, Y) random variables on $\mathcal{Z} = \mathbb{R}^D \times \mathbb{R}$,
- we want to estimate $\mathbb{E}(Y \mid X = x)$.

Density estimation,

- X random variable on $\mathcal{Z} = \mathbb{R}^D$,
- we want to estimate μ_X.

The boundaries between these categories are fuzzy:

- Regression allows to do classification through class scores.
- Density models allow to do classification thanks to Bayes' law.

etc.

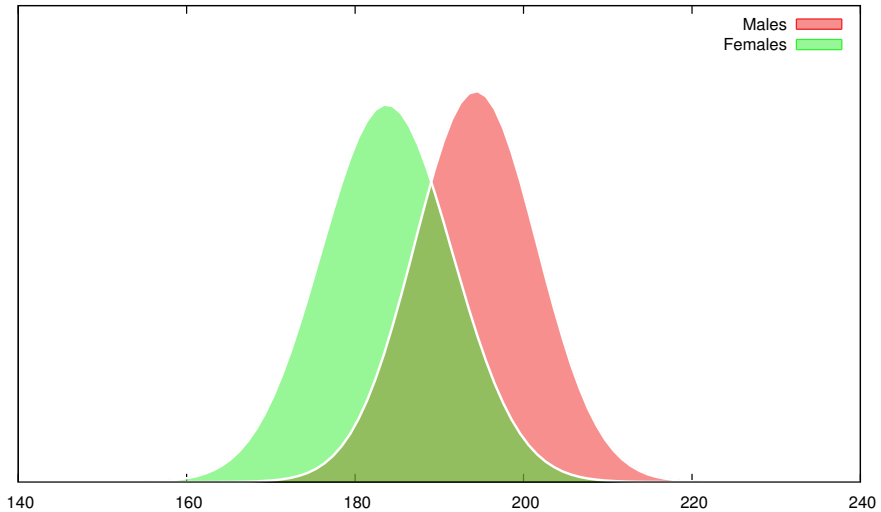
We call **generative** classification methods with an explicit data model, and **discriminative** the ones bypassing such a modeling .

We call **generative** classification methods with an explicit data model, and **discriminative** the ones bypassing such a modeling .

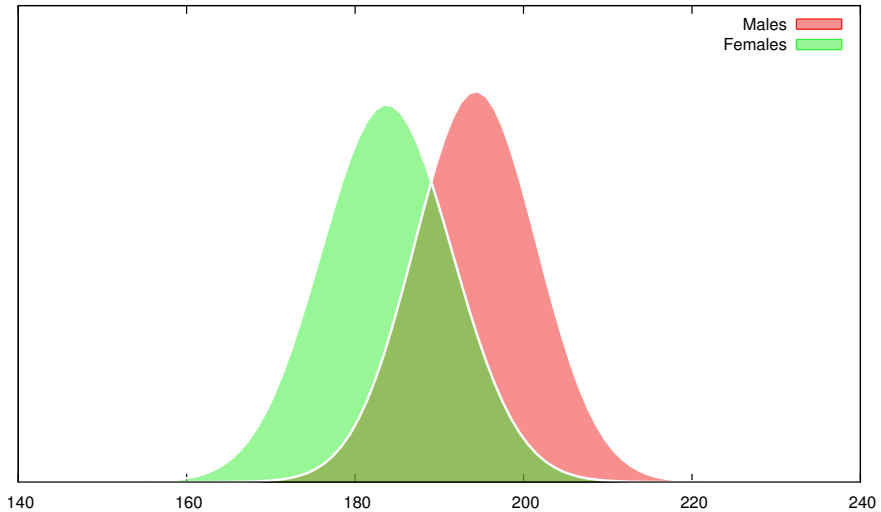
Example: Can we predict a Brazilian basketball player's gender G from his/her height H?

Females:	190	182	188	184	196	173	180	193	179	186	185	169
Males:	192	190	183	199	200	190	195	184	190	203	205	201

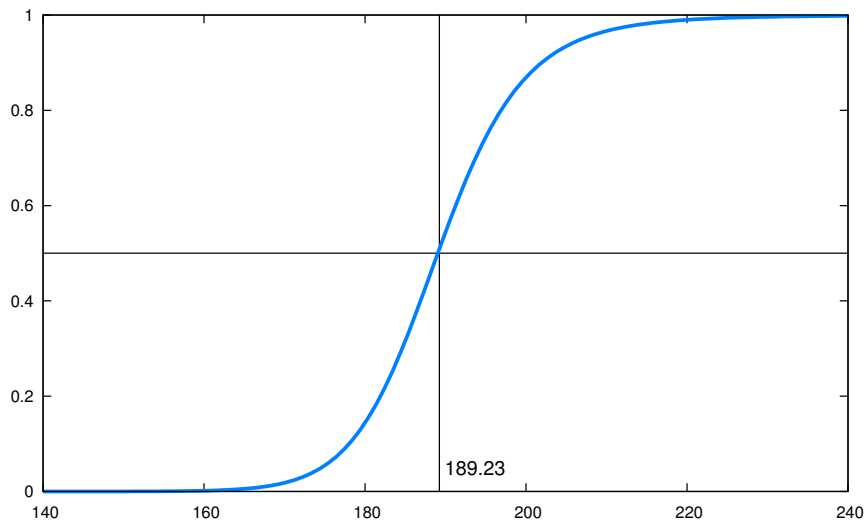
In the **generative** approach, we model $\mu_{H|G=g}(h)$



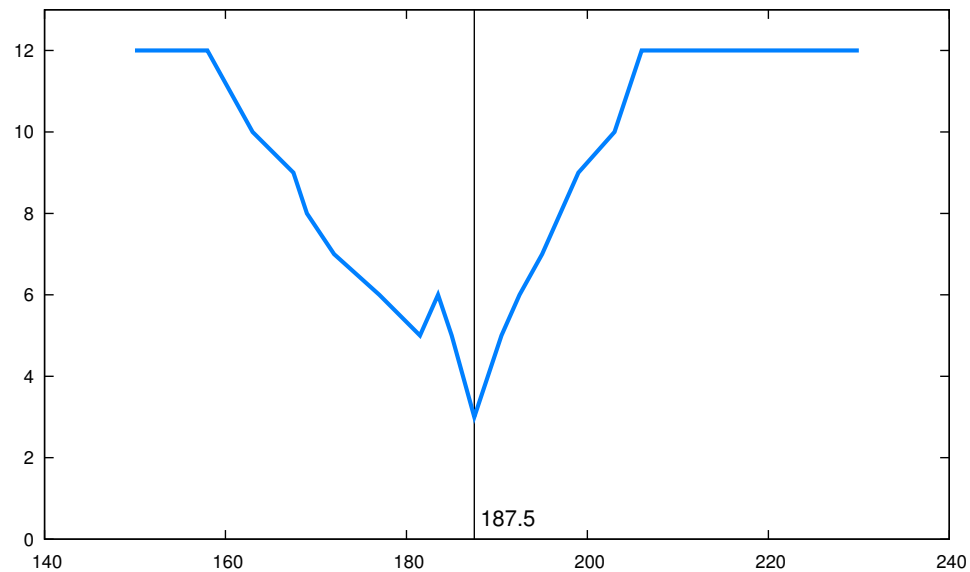
In the **generative** approach, we model $\mu_{H|G=g}(h)$



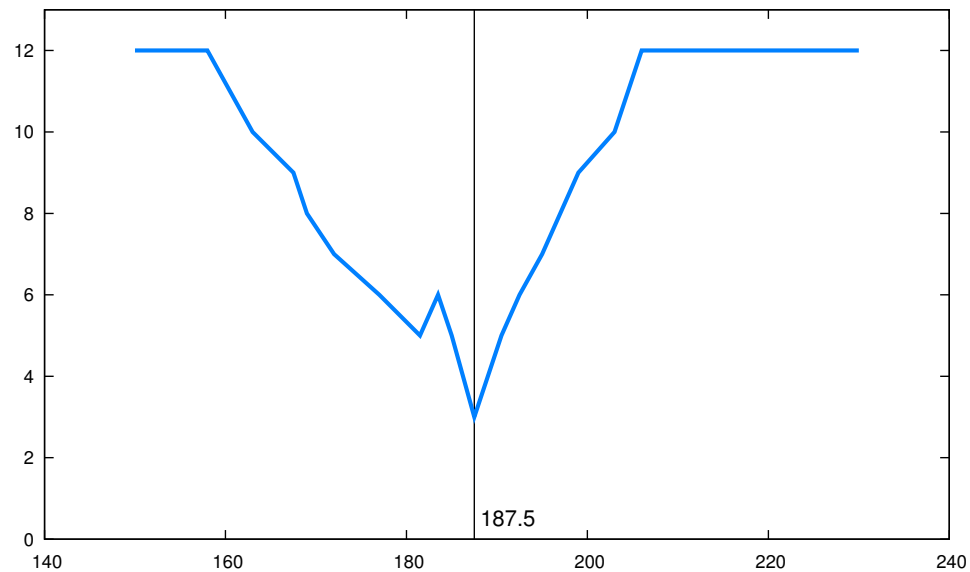
and use Bayes's law $P(G = g \mid H = h) = \frac{\mu_{H|G=g}(h)P(G=g)}{\mu_H(h)}$



In the **discriminative** approach we directly pick the threshold that works the best on the data:



In the **discriminative** approach we directly pick the threshold that works the best on the data:



Note that it is harder to design a confidence indicator.

Risk, empirical risk

Learning consists of finding in a set $\mathscr{F}$ of functionals a "good" f^* (or its parameters' values) usually defined through a loss

$$\ell : \mathscr{F} \times \mathscr{Z} \to \mathbb{R}$$

such that $\ell(f, z)$ increases with how wrong f is on z.

Learning consists of finding in a set $\mathscr{F}$ of functionals a "good" f^* (or its parameters' values) usually defined through a loss

$$\ell : \mathscr{F} \times \mathscr{Z} \to \mathbb{R}$$

such that $\ell(f, z)$ increases with how wrong f is on z. For instance

- for classification:
$$\ell(f, (x, y)) = \mathbf{1}_{\{f(x) \neq y\}},$$
- for regression:
$$\ell(f, (x, y)) = (f(x) - y)^2,$$
- for density estimation:
$$\ell(q, z) = -\log q(z).$$

Learning consists of finding in a set $\mathscr{F}$ of functionals a "good" f^* (or its parameters' values) usually defined through a loss

$$\ell : \mathscr{F} \times \mathscr{Z} \to \mathbb{R}$$

such that $\ell(f, z)$ increases with how wrong f is on z. For instance

- for classification:
$$\ell(f, (x, y)) = \mathbf{1}_{\{f(x) \neq y\}},$$
- for regression:
$$\ell(f, (x, y)) = (f(x) - y)^2,$$
- for density estimation:
$$\ell(q, z) = -\log q(z).$$

The loss may include additional terms related to f itself.

We are looking for an f with a small **expected risk**

$$R(f) = \mathbb{E}_Z\left(\ell(f, Z)\right),$$

which means that our learning procedure would ideally choose

$$f^* = \operatorname*{argmin}_{f \in \mathcal{F}} R(f).$$

We are looking for an f with a small **expected risk**

$$R(f) = \mathbb{E}_Z\left(\ell(f, Z)\right),$$

which means that our learning procedure would ideally choose

$$f^* = \operatorname*{argmin}_{f \in \mathscr{F}} R(f).$$

Although this quantity is unknown, if we have i.i.d. training samples

$$\mathscr{D} = \{Z_1, \ldots, Z_N\},$$

we can compute an estimate, the **empirical risk**:

$$\hat{R}(f; \mathscr{D}) = \hat{\mathbb{E}}_{\mathscr{D}}(\ell(f, Z)) = \frac{1}{N}\sum_{n=1}^{N} \ell(f, Z_n).$$

We have

$$\mathbb{E}_{Z_1,\ldots,Z_N}\left(\hat{R}(f;\mathscr{D})\right) \quad = \quad \mathbb{E}_{Z_1,\ldots,Z_N}\left(\frac{1}{N}\sum_{n=1}^{N}\ell(f,Z_n)\right)$$

We have

$$\begin{aligned}\mathbb{E}_{Z_1,\ldots,Z_N}\left(\hat{R}(f;\mathscr{D})\right) &= \mathbb{E}_{Z_1,\ldots,Z_N}\left(\frac{1}{N}\sum_{n=1}^{N}\ell(f,Z_n)\right)\\ &= \frac{1}{N}\sum_{n=1}^{N}\mathbb{E}_{Z_n}\left(\ell(f,Z_n)\right)\end{aligned}$$

We have

$$\begin{aligned}
\mathbb{E}_{Z_1,\ldots,Z_N}\left(\hat{R}(f;\mathscr{D})\right) &= \mathbb{E}_{Z_1,\ldots,Z_N}\left(\frac{1}{N}\sum_{n=1}^{N}\ell(f,Z_n)\right) \\
&= \frac{1}{N}\sum_{n=1}^{N}\mathbb{E}_{Z_n}\left(\ell(f,Z_n)\right) \\
&= \frac{1}{N}\sum_{n=1}^{N}\mathbb{E}_{Z}\left(\ell(f,Z)\right)
\end{aligned}$$

We have

$$\begin{aligned}
\mathbb{E}_{Z_1,\ldots,Z_N}\left(\hat{R}(f;\mathscr{D})\right) &= \mathbb{E}_{Z_1,\ldots,Z_N}\left(\frac{1}{N}\sum_{n=1}^{N}\ell(f,Z_n)\right) \\
&= \frac{1}{N}\sum_{n=1}^{N}\mathbb{E}_{Z_n}\left(\ell(f,Z_n)\right) \\
&= \frac{1}{N}\sum_{n=1}^{N}\mathbb{E}_{Z}\left(\ell(f,Z)\right) \\
&= \mathbb{E}_{Z}\left(\ell(f,Z)\right)
\end{aligned}$$

We have

$$\begin{aligned}
\mathbb{E}_{Z_1,\ldots,Z_N}\left(\hat{R}(f;\mathscr{D})\right) &= \mathbb{E}_{Z_1,\ldots,Z_N}\left(\frac{1}{N}\sum_{n=1}^{N}\ell(f,Z_n)\right)\\
&= \frac{1}{N}\sum_{n=1}^{N}\mathbb{E}_{Z_n}\left(\ell(f,Z_n)\right)\\
&= \frac{1}{N}\sum_{n=1}^{N}\mathbb{E}_{Z}\left(\ell(f,Z)\right)\\
&= \mathbb{E}_{Z}\left(\ell(f,Z)\right)\\
&= R(f).
\end{aligned}$$

We have

$$\begin{aligned}
\mathbb{E}_{Z_1,\ldots,Z_N}\left(\hat{R}(f;\mathscr{D})\right) &= \mathbb{E}_{Z_1,\ldots,Z_N}\left(\frac{1}{N}\sum_{n=1}^{N}\ell(f,Z_n)\right) \\
&= \frac{1}{N}\sum_{n=1}^{N}\mathbb{E}_{Z_n}\left(\ell(f,Z_n)\right) \\
&= \frac{1}{N}\sum_{n=1}^{N}\mathbb{E}_{Z}\left(\ell(f,Z)\right) \\
&= \mathbb{E}_{Z}\left(\ell(f,Z)\right) \\
&= R(f).
\end{aligned}$$

The empirical risk is an **unbiased estimator** of the expected risk.

Finally, given $\mathscr{D}$, $\mathscr{F}$, and ℓ, "learning" aims at computing

$$f^* = \operatorname*{argmin}_{f \in \mathscr{F}} \hat{R}(f; \mathscr{D}).$$

Finally, given $\mathscr{D}$, $\mathscr{F}$, and ℓ, "learning" aims at computing

$$f^* = \operatorname*{argmin}_{f \in \mathscr{F}} \hat{R}(f; \mathscr{D}).$$

- Can we bound $R(f)$ with $\hat{R}(f; \mathscr{D})$?

  Yes if f is not chosen using $\mathscr{D}$. Since the Z_n are independent, we just need to take into account the variance of $\hat{R}(f; \mathscr{D})$.

Finally, given $\mathscr{D}$, $\mathscr{F}$, and ℓ, "learning" aims at computing

$$f^* = \operatorname*{argmin}_{f \in \mathscr{F}} \hat{R}(f; \mathscr{D}).$$

- Can we bound $R(f)$ with $\hat{R}(f; \mathscr{D})$?

  Yes if f is not chosen using $\mathscr{D}$. Since the Z_n are independent, we just need to take into account the variance of $\hat{R}(f; \mathscr{D})$.

- Can we bound $R(f^*)$ with $\hat{R}(f^*; \mathscr{D})$?

  Unfortunately not simply, and not without additional constraints on $\mathscr{F}$.

Finally, given $\mathscr{D}$, $\mathscr{F}$, and ℓ, "learning" aims at computing

$$f^* = \operatorname*{argmin}_{f \in \mathscr{F}} \hat{R}(f; \mathscr{D}).$$

- Can we bound $R(f)$ with $\hat{R}(f; \mathscr{D})$?

  Yes if f is not chosen using $\mathscr{D}$. Since the Z_n are independent, we just need to take into account the variance of $\hat{R}(f; \mathscr{D})$.

- Can we bound $R(f^*)$ with $\hat{R}(f^*; \mathscr{D})$?

  Unfortunately not simply, and not without additional constraints on $\mathscr{F}$.

  For instance if $|\mathscr{F}| = 1$, we can!

Note that in practice, we call "loss" both the functional

$$\ell : \mathscr{F} \times \mathscr{Z} \to \mathbb{R}$$

and the empirical risk minimized during training

$$\mathscr{L}(f) = \frac{1}{N} \sum_{n=1}^{N} \ell(f, z_n).$$

Over and under-fitting, intuition

You want to hire someone, and you evaluate candidates by asking them ten technical yes/no questions.

You want to hire someone, and you evaluate candidates by asking them ten technical yes/no questions.

Would you feel confident if you interviewed one candidate and he makes a perfect score?

You want to hire someone, and you evaluate candidates by asking them ten technical yes/no questions.

Would you feel confident if you interviewed one candidate and he makes a perfect score?

What about interviewing ten candidates and picking the best?

You want to hire someone, and you evaluate candidates by asking them ten technical yes/no questions.

Would you feel confident if you interviewed one candidate and he makes a perfect score?

What about interviewing ten candidates and picking the best?

What about one thousand?

With

$$Q_k^n \sim \mathscr{B}(0.5), \;\; n = 1, \ldots, 1000, \;\; k = 1, \ldots, 10,$$

independent, we have

$$\forall n, \;\; P(\forall k, Q_k^n = 1) = \frac{1}{1024}$$

and

$$P(\exists n, \forall k, Q_k^n = 1) \simeq 0.62.$$

With

$$Q_k^n \sim \mathscr{B}(0.5), \;\; n = 1, \ldots, 1000, \;\; k = 1, \ldots, 10,$$

independent, we have

$$\forall n, \;\; P(\forall k, Q_k^n = 1) = \frac{1}{1024}$$

and

$$P(\exists n, \forall k, Q_k^n = 1) \simeq 0.62.$$

There is 62% chance that among $1,000$ candidates answering completely at random, one will score perfectly.

With

$$Q_k^n \sim \mathscr{B}(0.5), \;\; n = 1, \ldots, 1000, \;\; k = 1, \ldots, 10,$$

independent, we have

$$\forall n, \;\; P(\forall k, Q_k^n = 1) = \frac{1}{1024}$$

and

$$P(\exists n, \forall k, Q_k^n = 1) \simeq 0.62.$$

There is 62% chance that among $1,000$ candidates answering completely at random, one will score perfectly.

It is quite intuitive that selecting a candidate based on a statistical estimator biases the said estimator for that candidate.

Over and under-fitting, capacity. K-nearest-neighbors

A simple classification procedure is the "K-nearest neighbors."

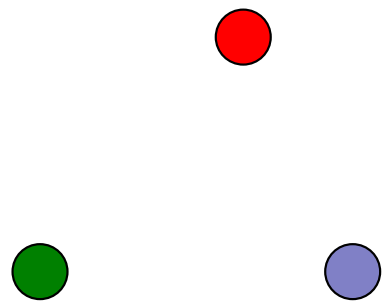
Given

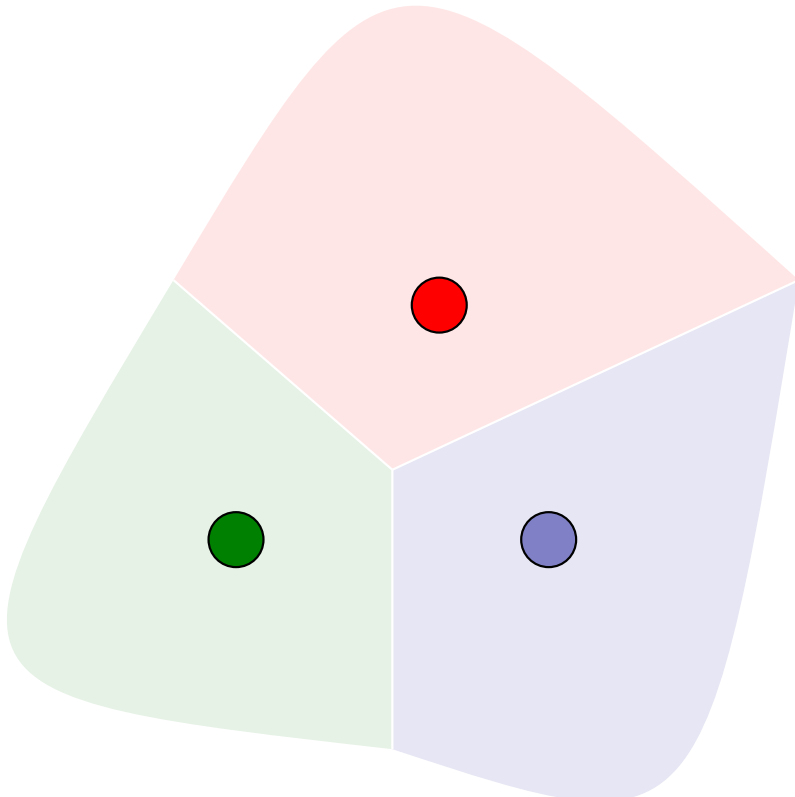
$$(x_n, y_n) \in \mathbb{R}^D \times \{1, \ldots, C\}, \ n = 1, \ldots, N$$

to predict the y associated to a new x, take the y_n of the closest x_n:

$$\begin{aligned} n^*(x) &= \operatorname*{argmin}_n \|x_n - x\| \\ f^*(x) &= y_{n^*(x)}. \end{aligned}$$

This recipe corresponds to $K = 1$, and makes the empirical training error zero.





$K = 1$

Under mild assumptions of regularities of $\mu_{X,Y}$, for $N \to \infty$ the asymptotic error rate of the 1-NN is less than twice the (optimal!) Bayes' Error rate.

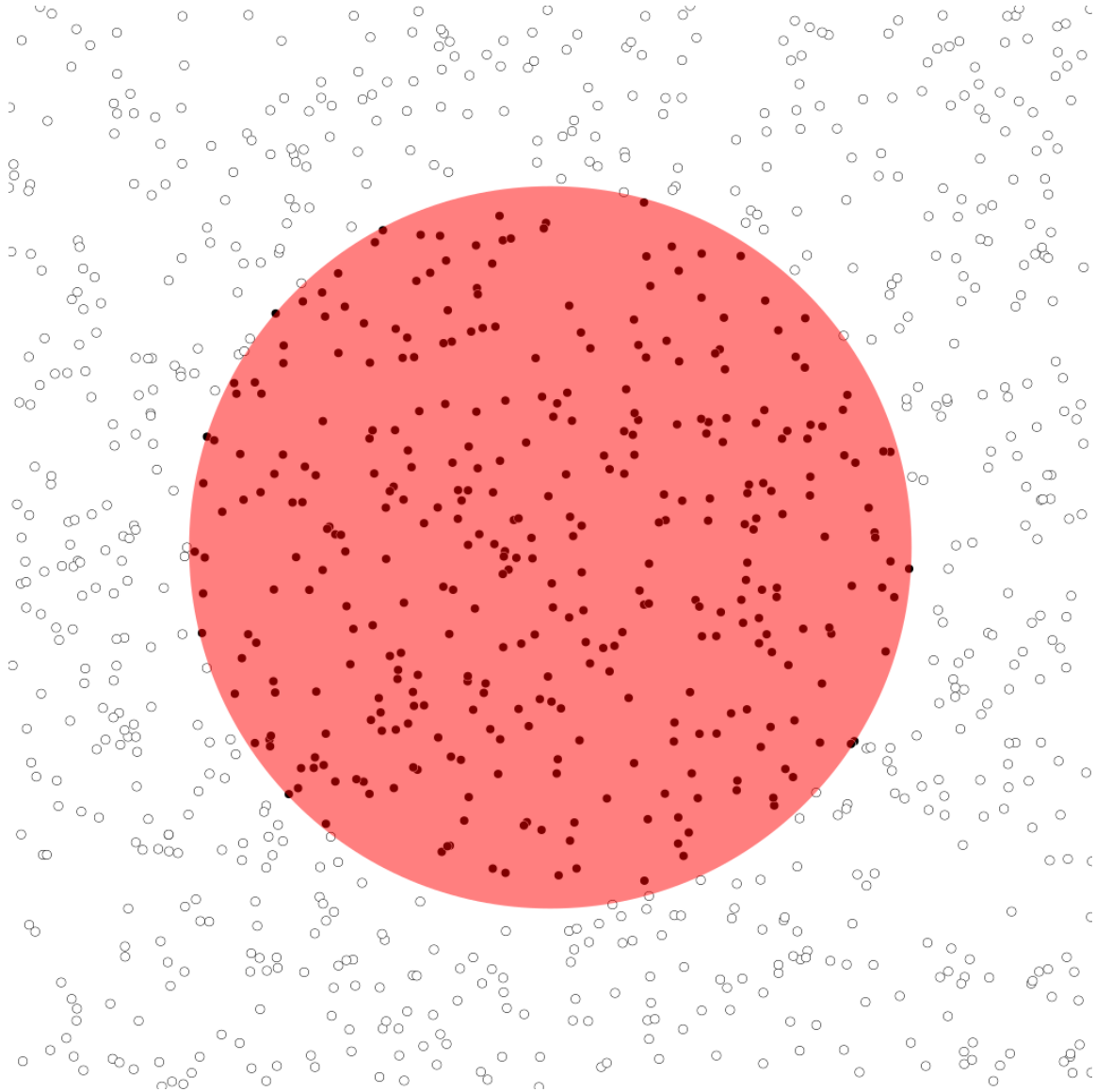
Under mild assumptions of regularities of $\mu_{X,Y}$, for $N \to \infty$ the asymptotic error rate of the 1-NN is less than twice the (optimal!) Bayes' Error rate.

It can be made more stable by looking at the $K > 1$ closest training points, and taking the majority vote.

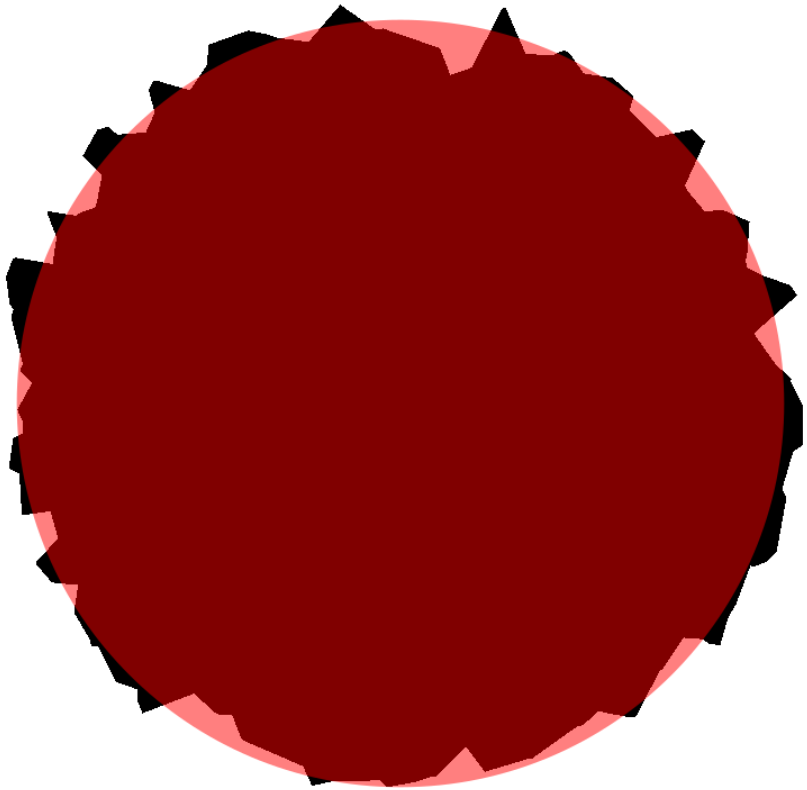
Under mild assumptions of regularities of $\mu_{X,Y}$, for $N \to \infty$ the asymptotic error rate of the 1-NN is less than twice the (optimal!) Bayes' Error rate.

It can be made more stable by looking at the $K > 1$ closest training points, and taking the majority vote.

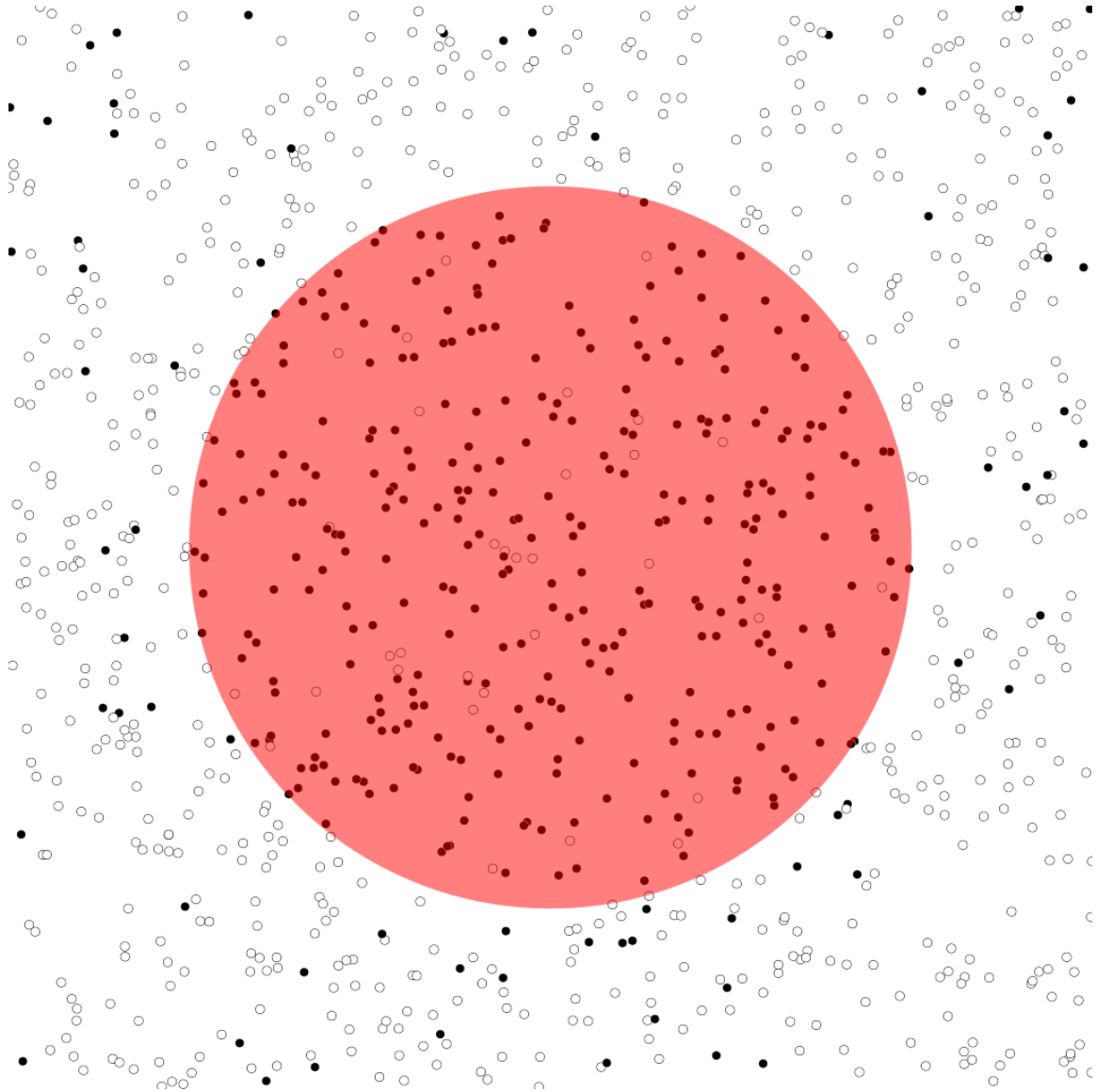
If we let also $K \to \infty$ "not too fast", the error rate is the (optimal!) Bayes' Error rate.



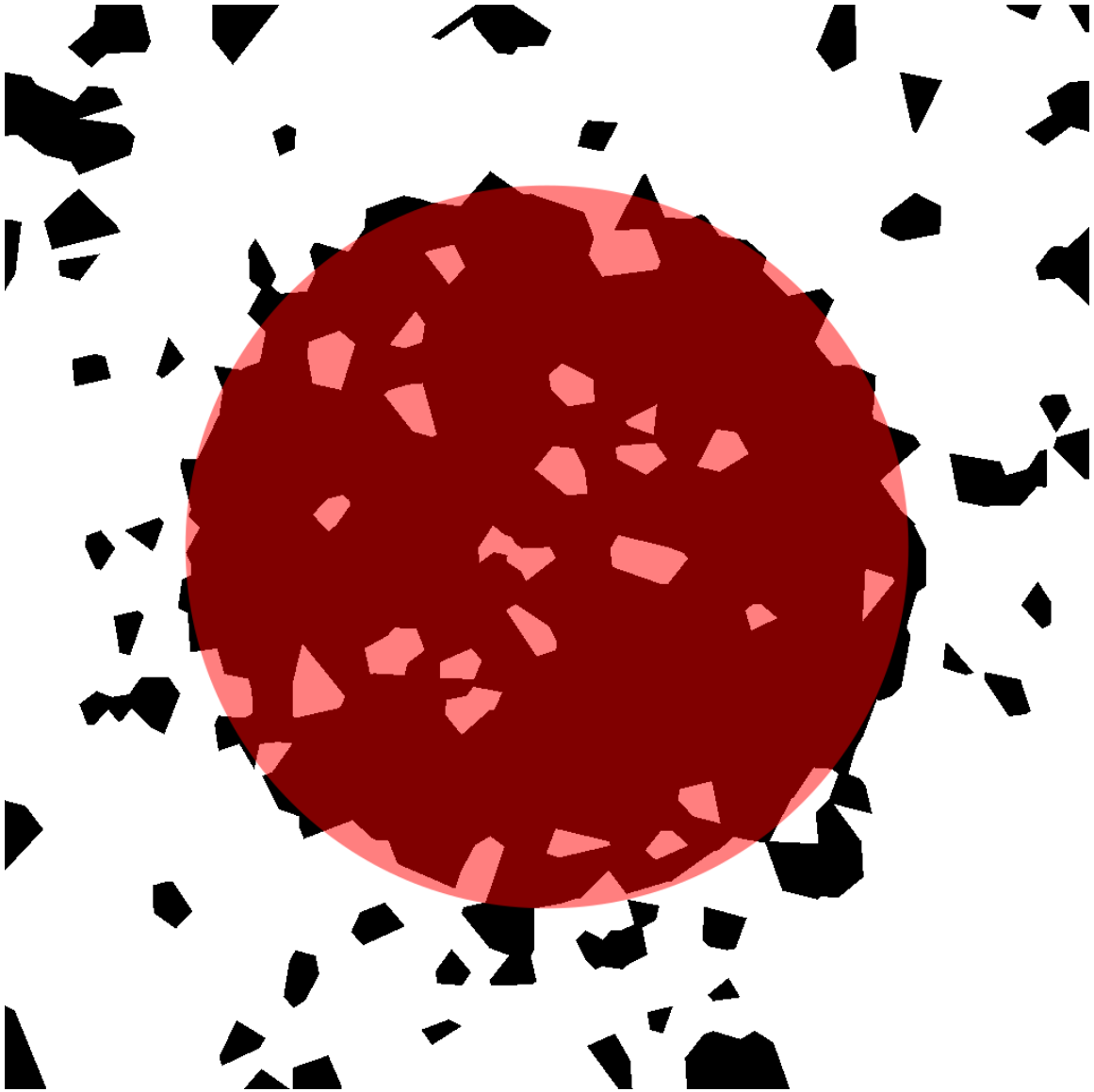
Training set



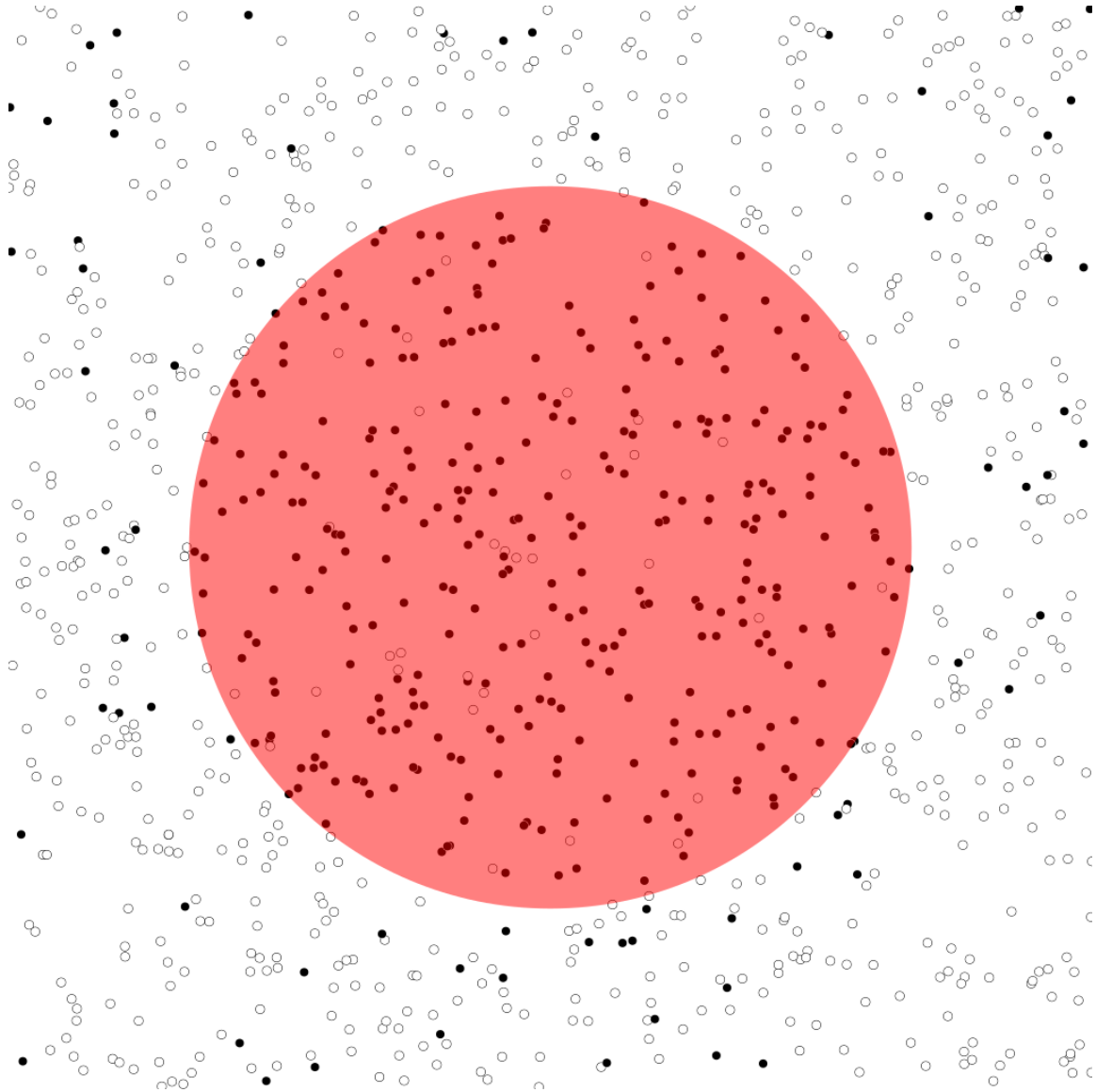
Prediction (K=1)



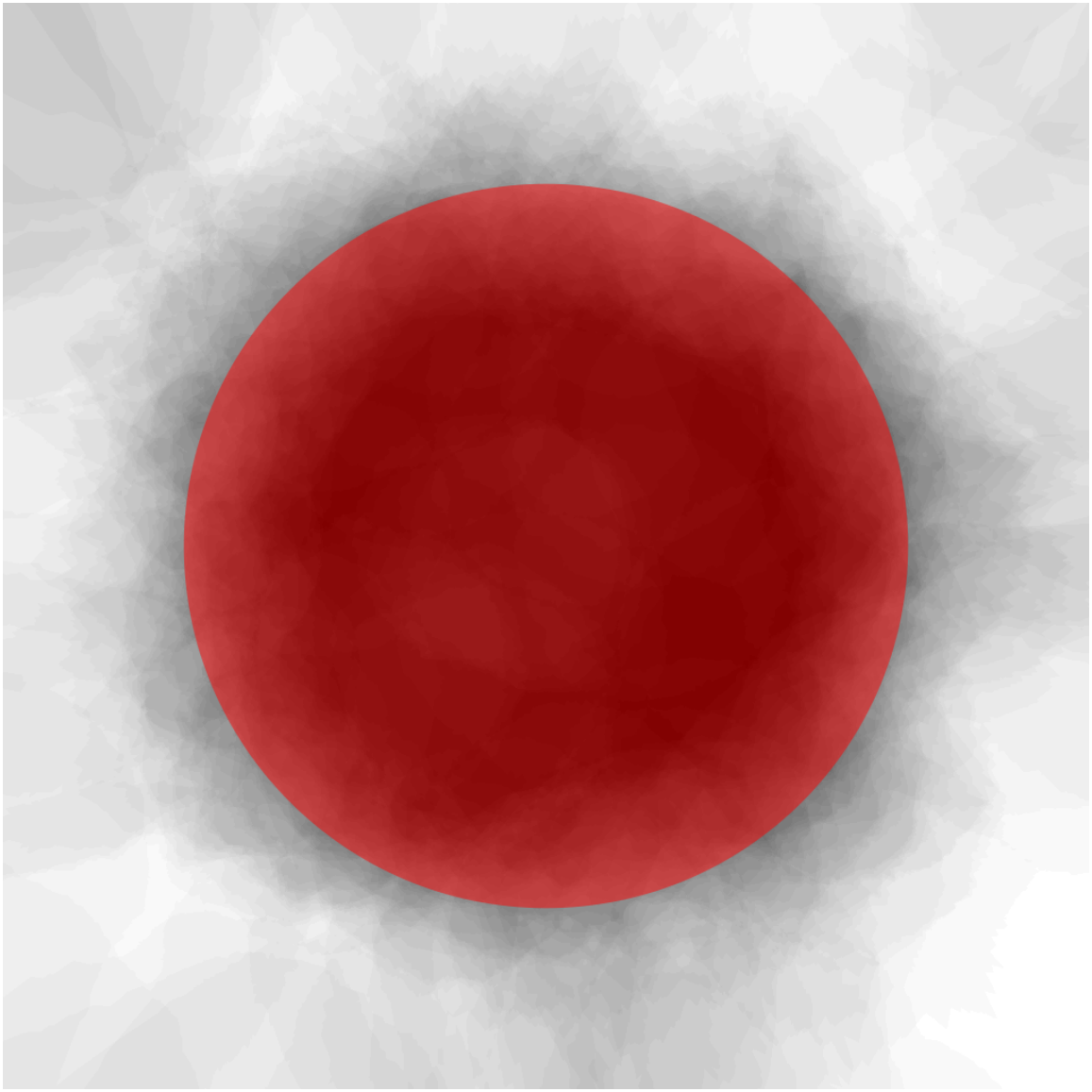
Training set



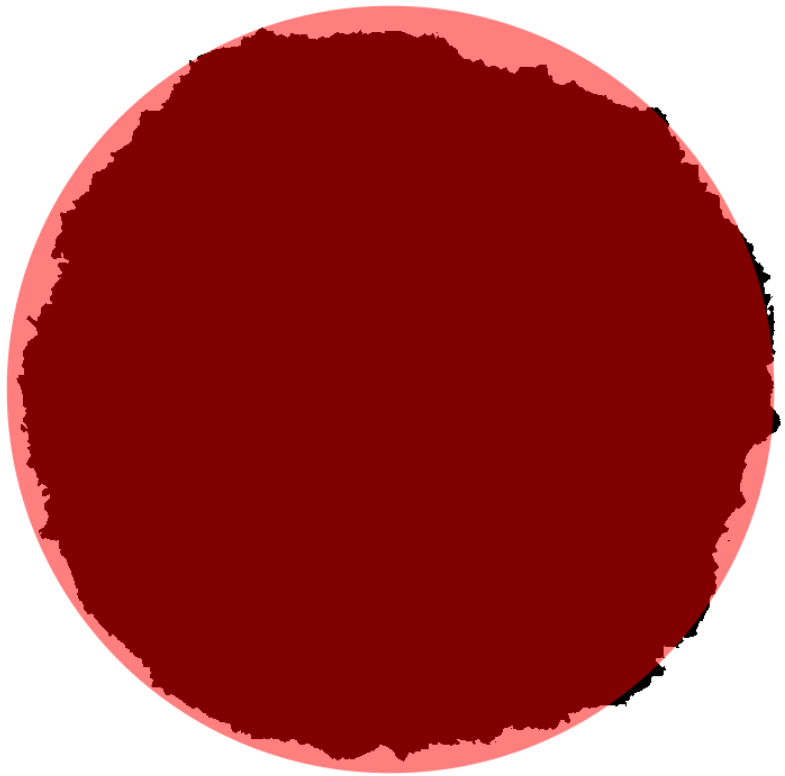
Prediction (K=1)



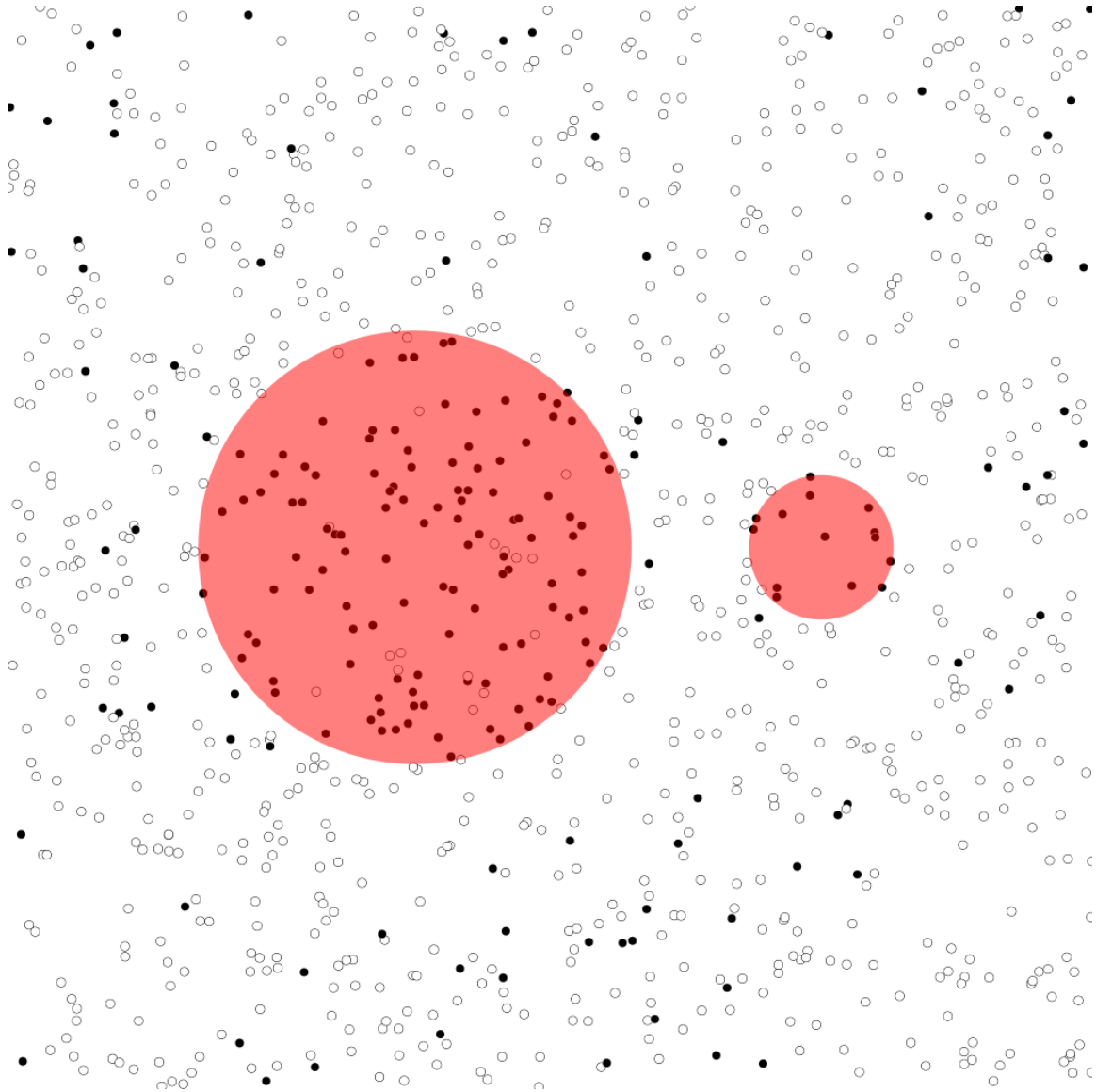
Training set



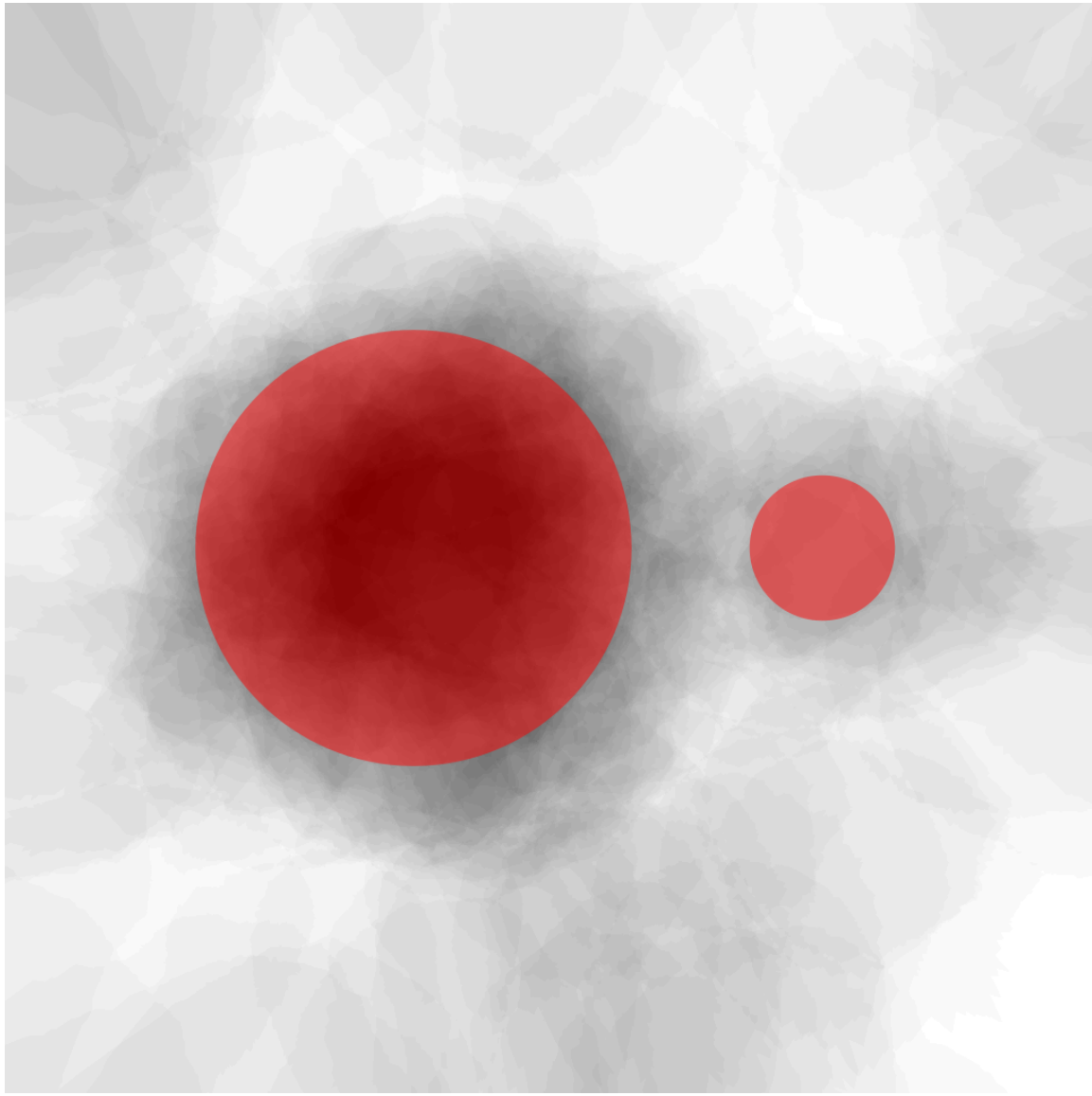
Votes (K=51)



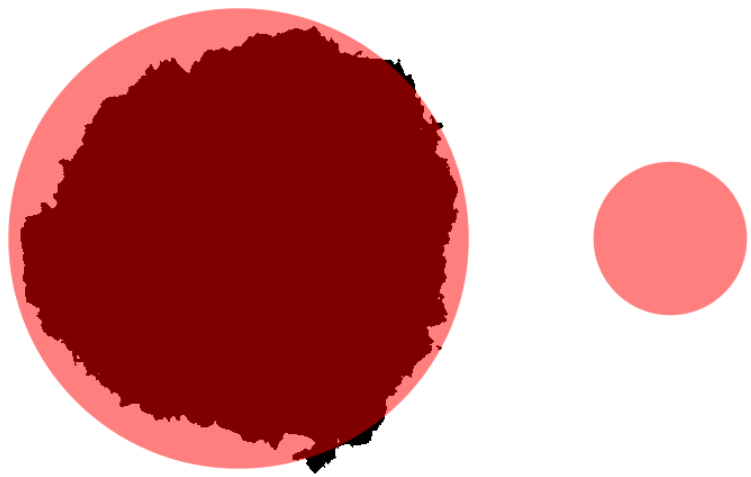
Prediction (K=51)



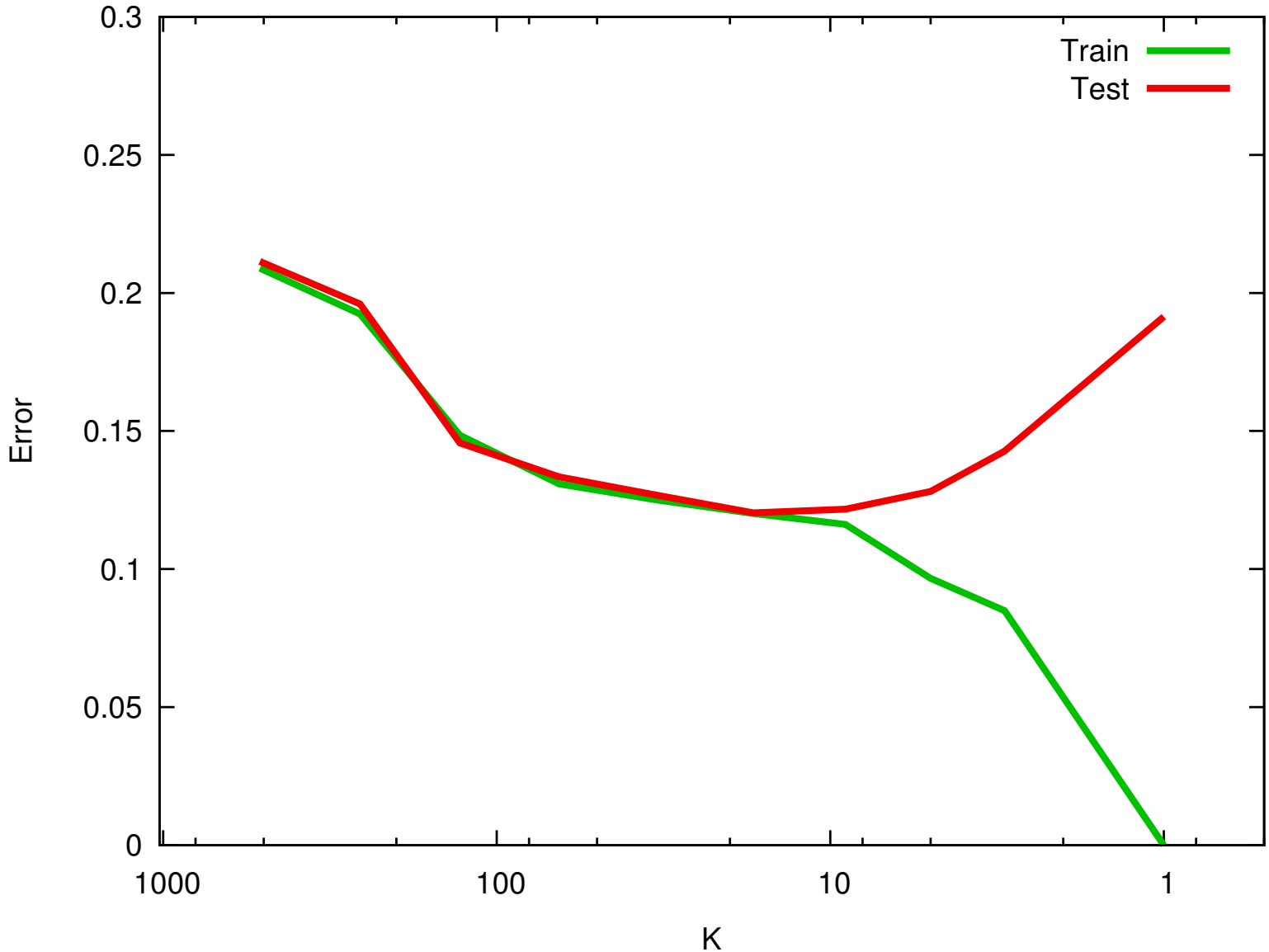
Training set

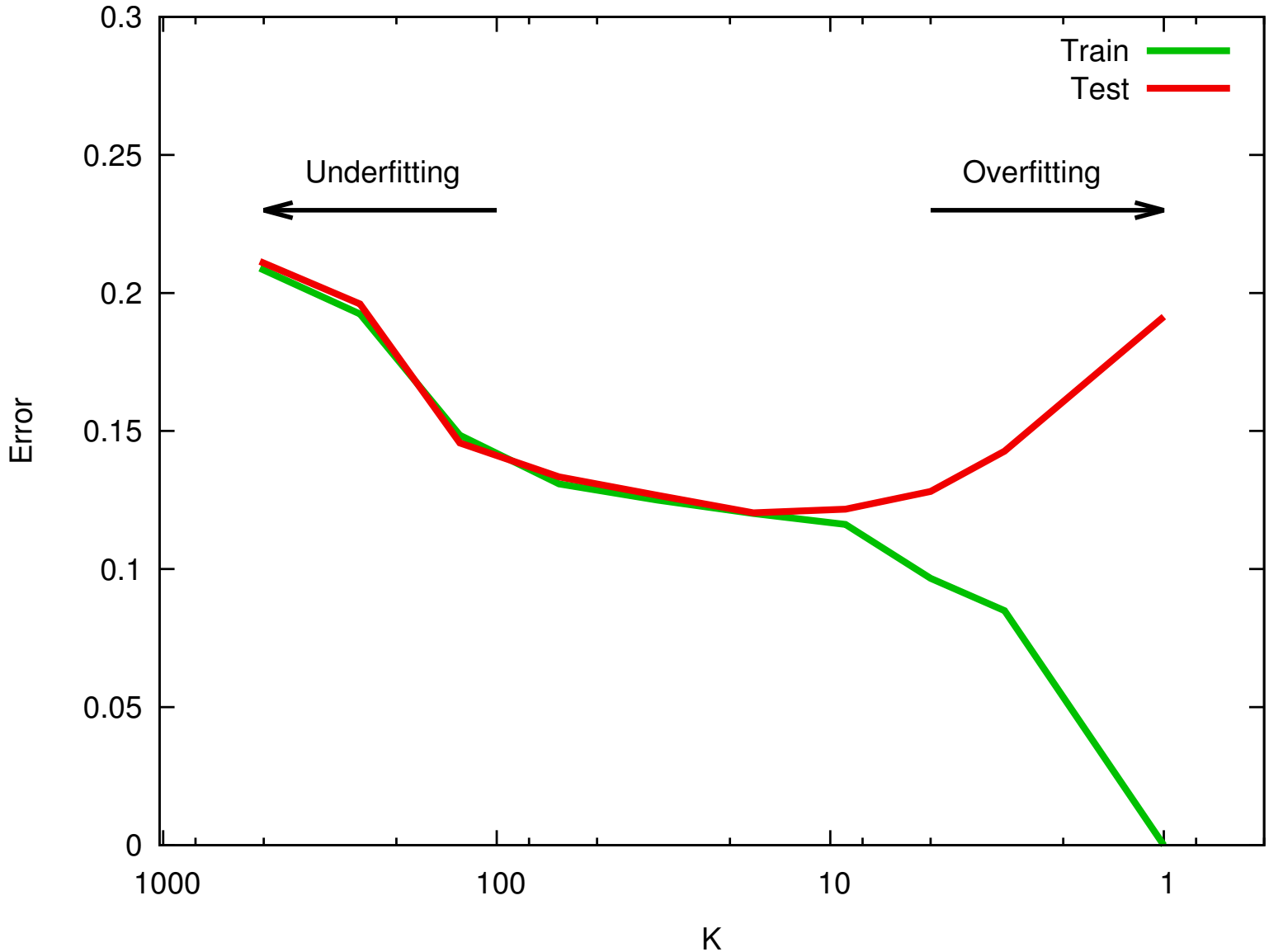


Votes (K=51)



Prediction (K=51)


0.3
0.25
0.2
0.15
0.1
0.05
0
Error
1000
100
10
1
K
Train
Test


0.3
0.25
0.2
0.15
0.1
0.05
0
Error
1000
100
10
1
K
Train
Test
Underfitting
Overfitting

Over and under-fitting, capacity, polynomials

Consider a polynomial model

$$\forall x, \alpha_0, \ldots, \alpha_D \in \mathbb{R}, \ f(x; \alpha) = \sum_{d=0}^{D} \alpha_d x^d.$$

Consider a polynomial model

$$\forall x, \alpha_0, \ldots, \alpha_D \in \mathbb{R}, \ f(x; \alpha) = \sum_{d=0}^{D} \alpha_d x^d.$$

and training points $(x_n, y_n) \in \mathbb{R}^2, n = 1, \ldots, N$, minimize the quadratic loss

$$\mathscr{L}(\alpha) = \sum_n (f(x_n; \alpha) - y_n)^2$$

Consider a polynomial model

$$\forall x, \alpha_0, \ldots, \alpha_D \in \mathbb{R}, \ f(x; \alpha) = \sum_{d=0}^{D} \alpha_d x^d.$$

and training points $(x_n, y_n) \in \mathbb{R}^2, n = 1, \ldots, N$, minimize the quadratic loss

$$\begin{aligned} \mathscr{L}(\alpha) &= \sum_n (f(x_n; \alpha) - y_n)^2 \\ &= \sum_n \left(\sum_{d=0}^{D} \alpha_d x_n^d - y_n \right)^2 \end{aligned}$$

Consider a polynomial model

$$\forall x, \alpha_0, \ldots, \alpha_D \in \mathbb{R}, \ f(x; \alpha) = \sum_{d=0}^{D} \alpha_d x^d.$$

and training points $(x_n, y_n) \in \mathbb{R}^2, n = 1, \ldots, N$, minimize the quadratic loss

$$\begin{aligned}
\mathscr{L}(\alpha) &= \sum_n (f(x_n; \alpha) - y_n)^2 \\
&= \sum_n \left(\sum_{d=0}^{D} \alpha_d x_n^d - y_n \right)^2 \\
&= \left\| \begin{pmatrix} x_1^0 & \ldots & x_1^D \\ \vdots & & \vdots \\ x_N^0 & \ldots & x_N^D \end{pmatrix} \begin{pmatrix} \alpha_0 \\ \vdots \\ \alpha_D \end{pmatrix} - \begin{pmatrix} y_1 \\ \vdots \\ y_N \end{pmatrix} \right\|^2.
\end{aligned}$$

Consider a polynomial model

$$\forall x, \alpha_0, \ldots, \alpha_D \in \mathbb{R}, \ f(x; \alpha) = \sum_{d=0}^{D} \alpha_d x^d.$$

and training points $(x_n, y_n) \in \mathbb{R}^2, n = 1, \ldots, N$, minimize the quadratic loss

$$\begin{aligned}
\mathscr{L}(\alpha) &= \sum_n (f(x_n; \alpha) - y_n)^2 \\
&= \sum_n \left(\sum_{d=0}^{D} \alpha_d x_n^d - y_n \right)^2 \\
&= \left\| \begin{pmatrix} x_1^0 & \ldots & x_1^D \\ \vdots & & \vdots \\ x_N^0 & \ldots & x_N^D \end{pmatrix} \begin{pmatrix} \alpha_0 \\ \vdots \\ \alpha_D \end{pmatrix} - \begin{pmatrix} y_1 \\ \vdots \\ y_N \end{pmatrix} \right\|^2.
\end{aligned}$$

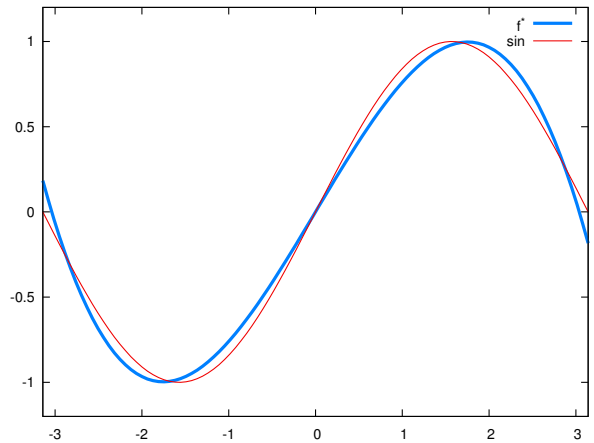
This is a standard quadratic problem, for which we have efficient algorithms.

$$\underset{\alpha}{\text{argmin}} \left\| \begin{pmatrix} x_1^0 & \dots & x_1^D \\ \vdots & & \vdots \\ x_N^0 & \dots & x_N^D \end{pmatrix} \begin{pmatrix} \alpha_0 \\ \vdots \\ \alpha_D \end{pmatrix} - \begin{pmatrix} y_1 \\ \vdots \\ y_N \end{pmatrix} \right\|^2$$

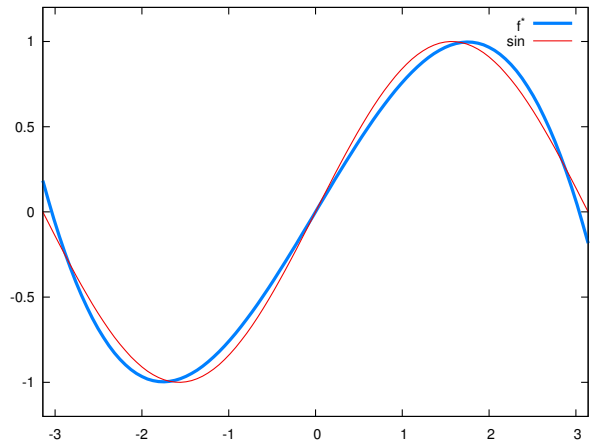
```
def fit_polynomial(D, x, y):
    N = x.size(0)

    X = Tensor(N, D + 1)
    for d in range(D + 1):
        X[:,d] = x.pow(d)

    Y = y.view(N, 1)

    # LAPACK's GEneralized Least-Square
    alpha, _ = torch.gels(Y, X)

    # unclear why we need narrow here (torch 0.3.1)
    return alpha.narrow(0, 0, D + 1)
```

```
D, N = 4, 100
x = torch.linspace(-math.pi, math.pi, N)
y = x.sin()

alpha = fit_polynomial(D, x, y)

X = Tensor(N, D + 1)
for d in range(D + 1):
    X[:,d] = x.pow(d)
z = X.mm(alpha).view(-1)

for k in range(N): print(x[k], y[k], z[k])
```

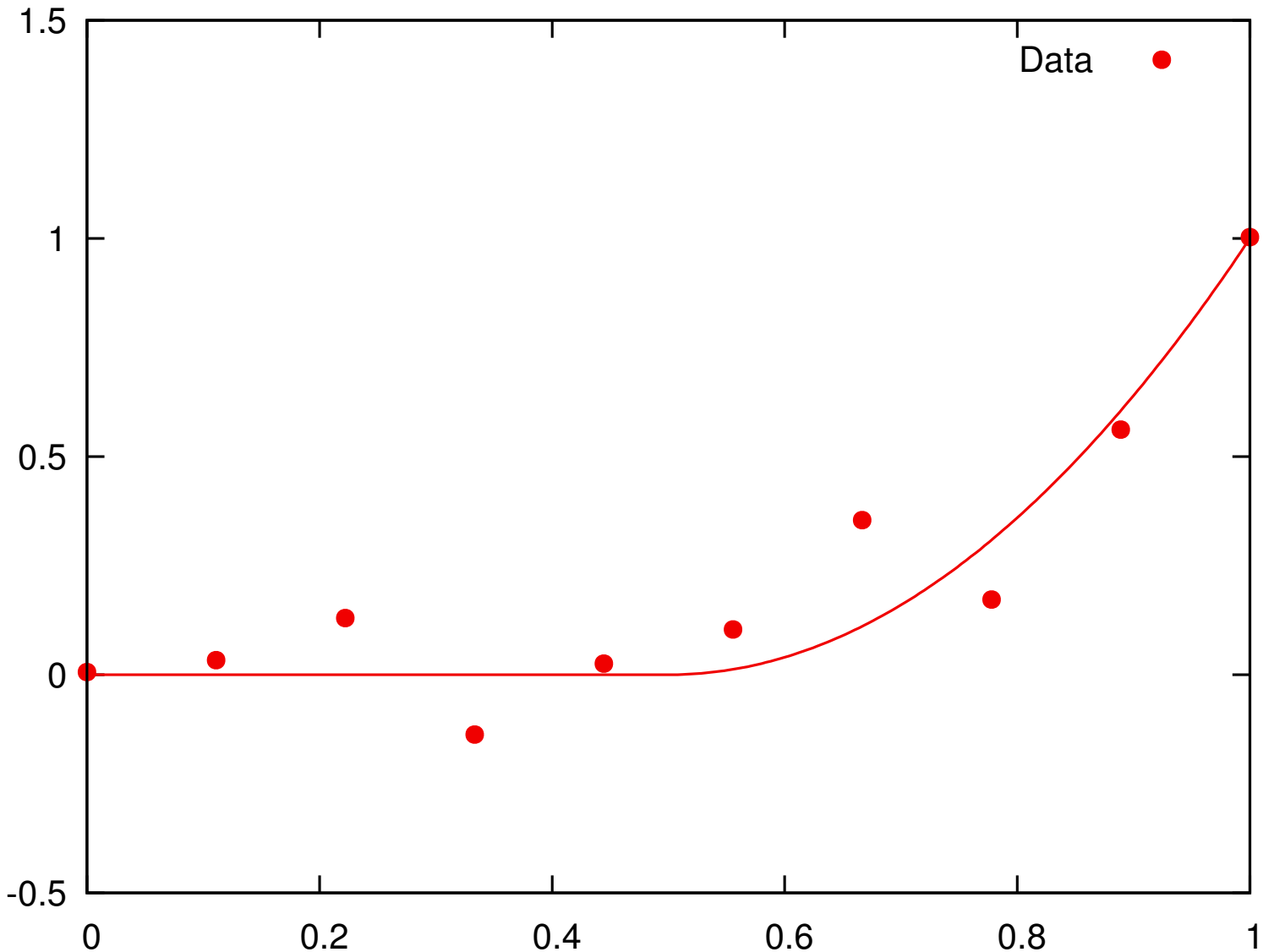
```
D, N = 4, 100
x = torch.linspace(-math.pi, math.pi, N)
y = x.sin()

alpha = fit_polynomial(D, x, y)

X = Tensor(N, D + 1)
for d in range(D + 1):
    X[:,d] = x.pow(d)
z = X.mm(alpha).view(-1)

for k in range(N): print(x[k], y[k], z[k])
```

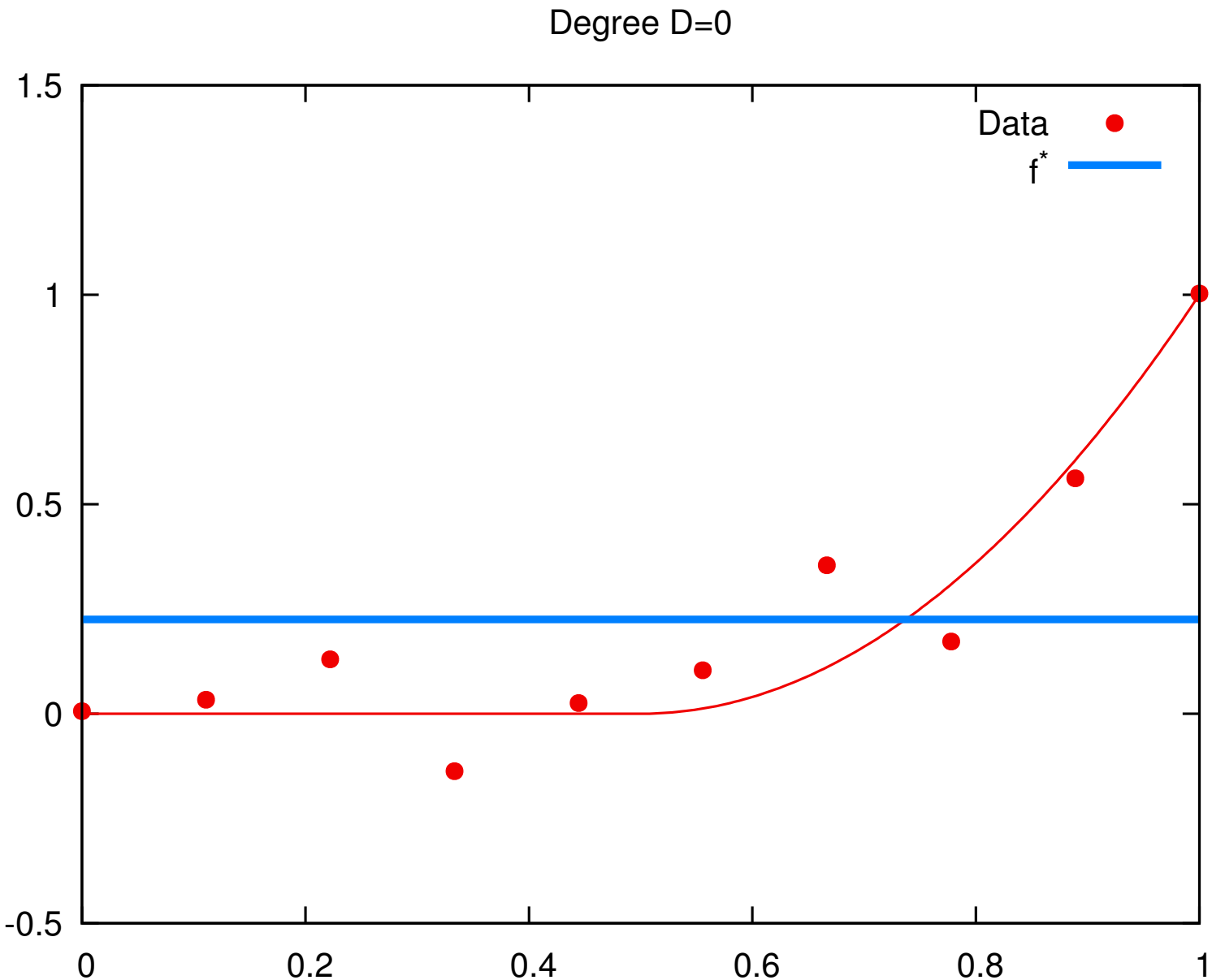


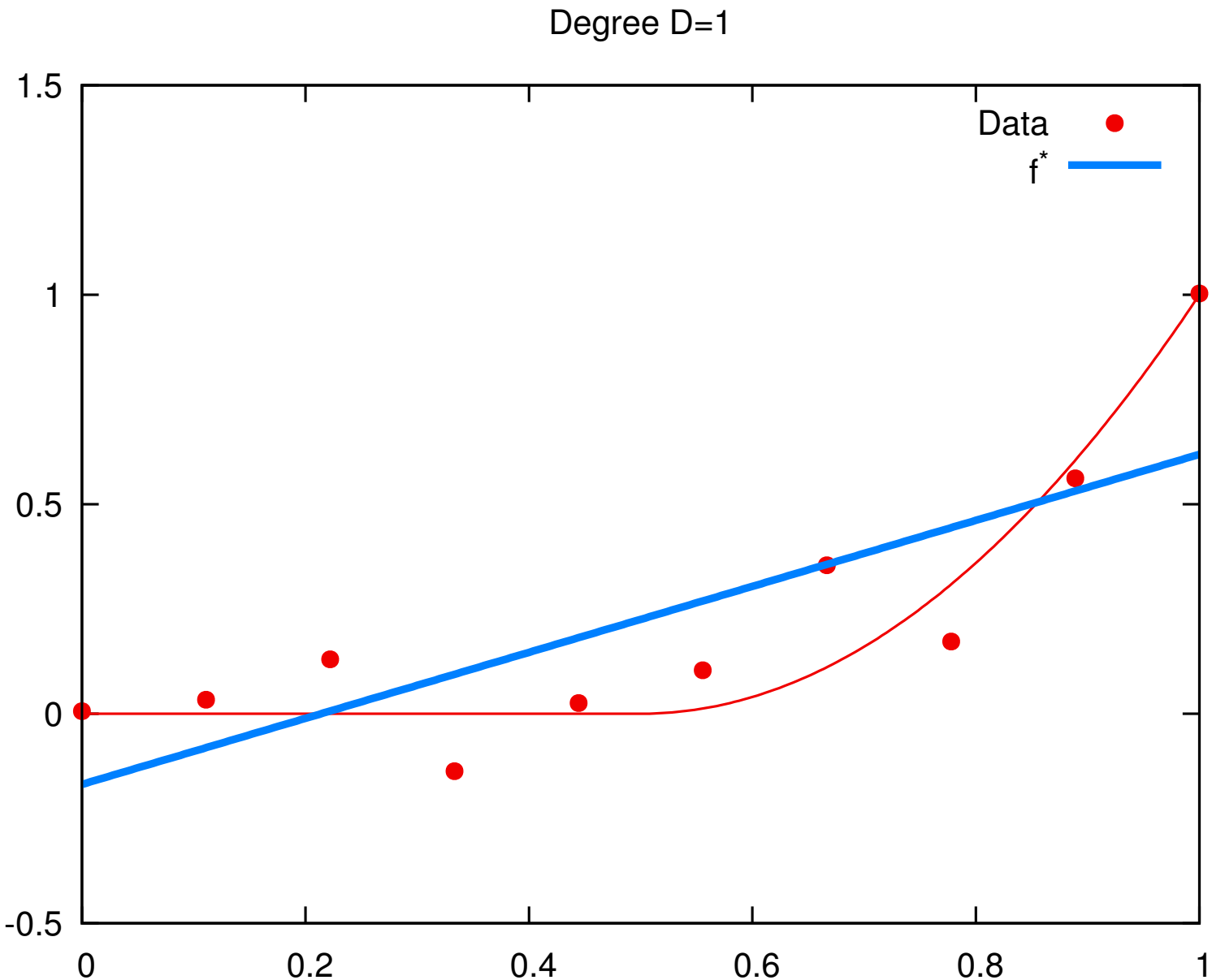
```
D, N = 4, 100
x = torch.linspace(-math.pi, math.pi, N)
y = x.sin()

alpha = fit_polynomial(D, x, y)

X = Tensor(N, D + 1)
for d in range(D + 1):
    X[:,d] = x.pow(d)
z = X.mm(alpha).view(-1)

for k in range(N): print(x[k], y[k], z[k])
```

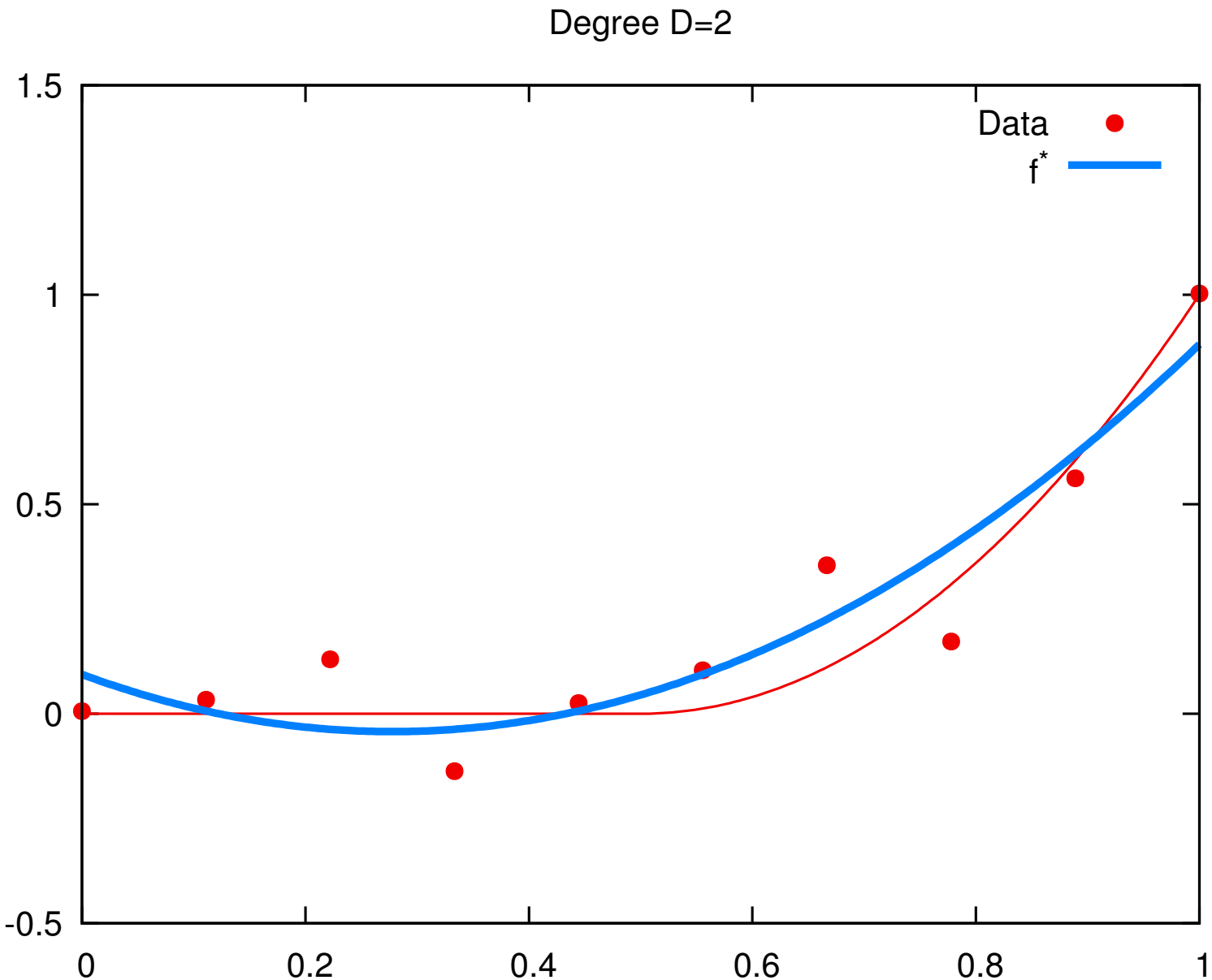


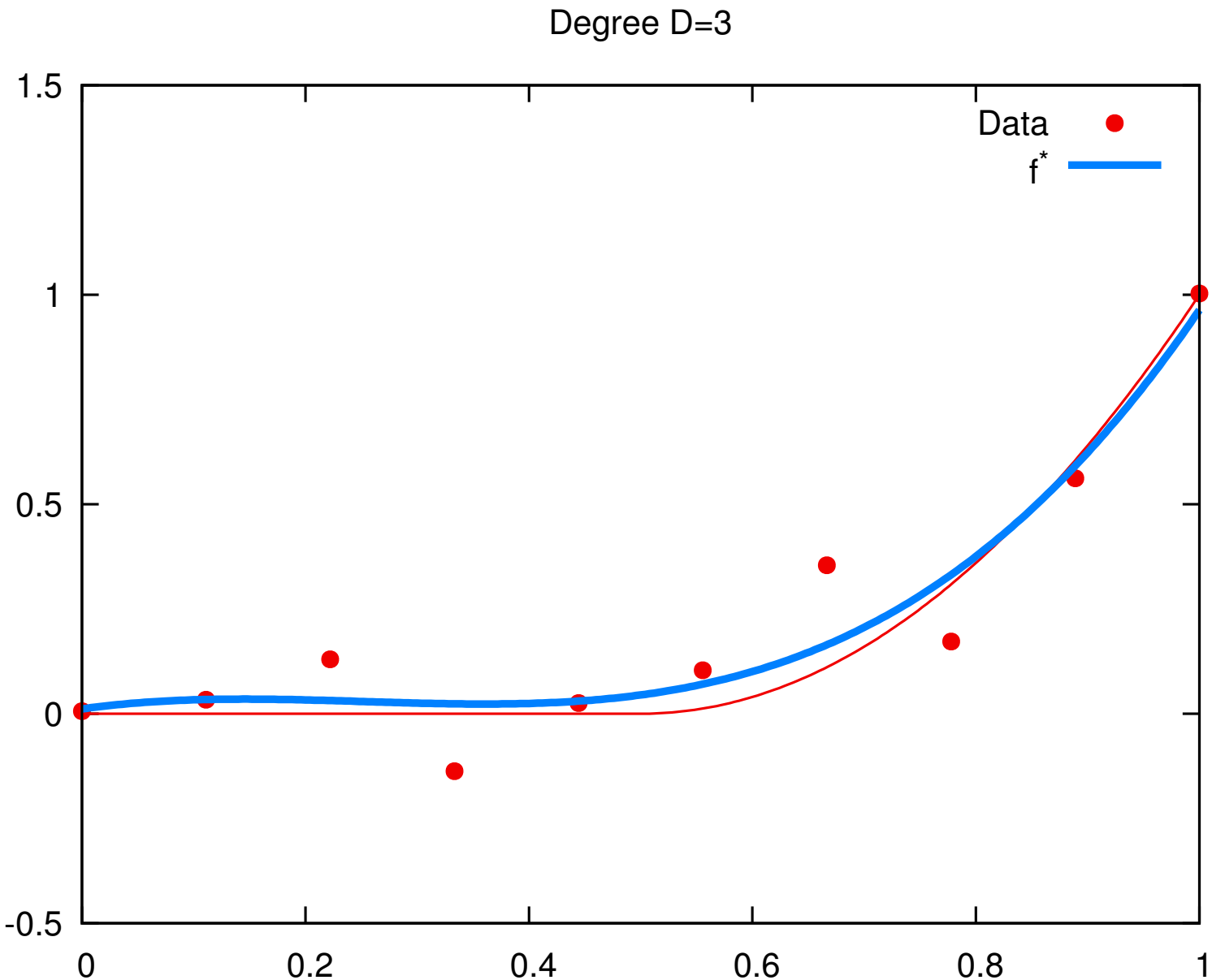
We can use that model on a synthetic example with noisy training points, to illustrate how the prediction changes when we increase the degree or the regularization.

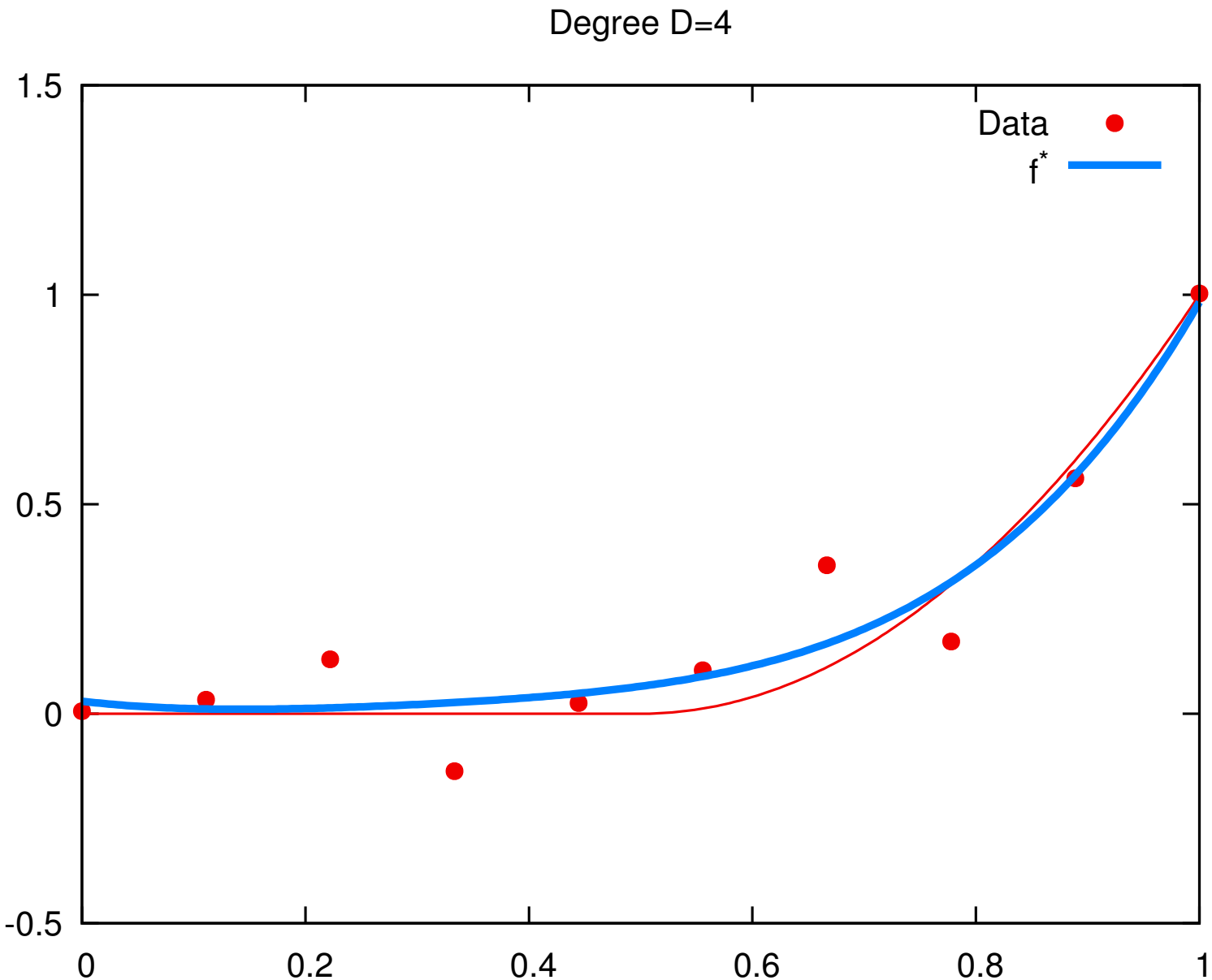

Data
1.5
1
0.5
0
-0.5
0
0.2
0.4
0.6
0.8
1

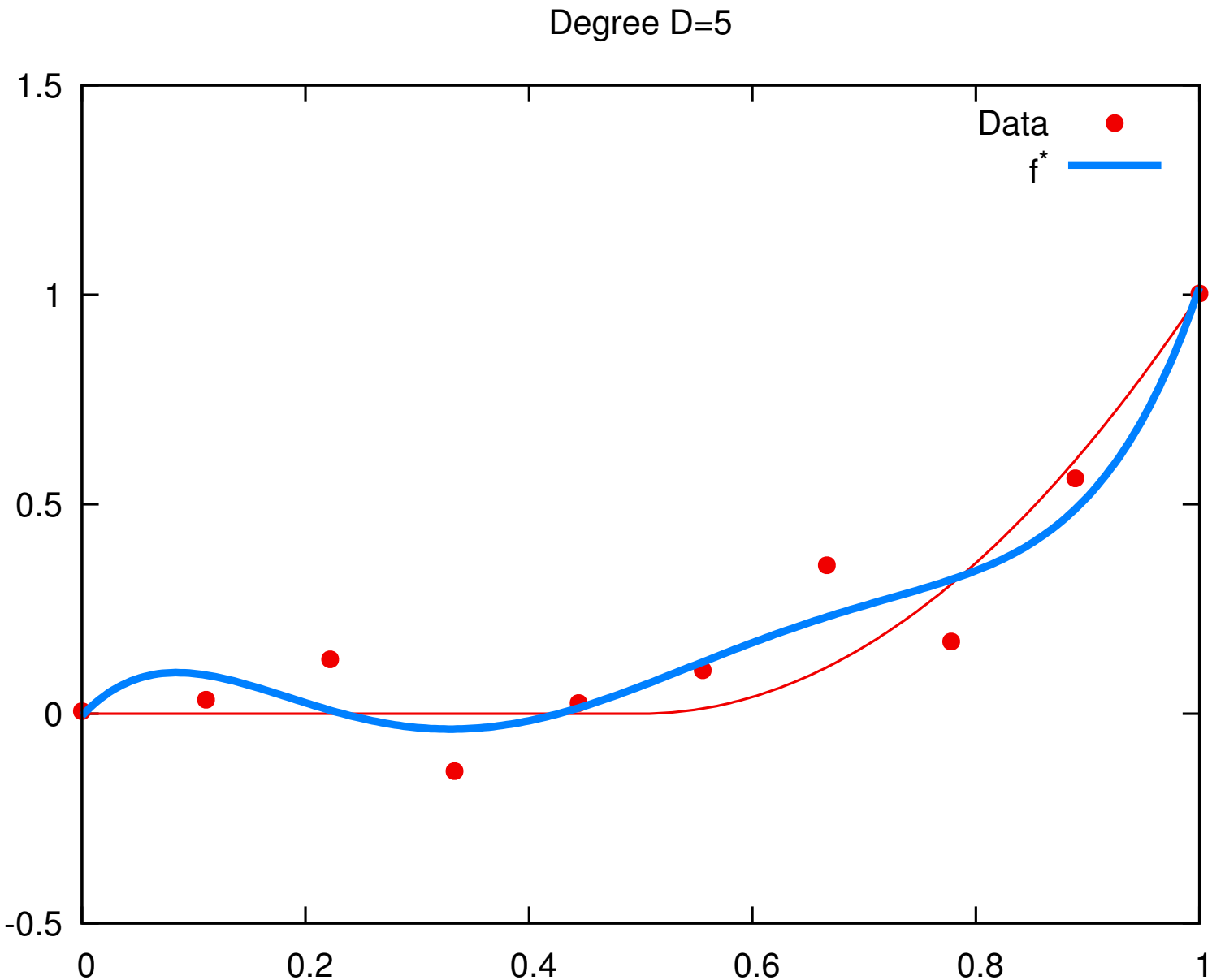

Degree D=0
Data
f*
1.5
1
0.5
0
-0.5
0
0.2
0.4
0.6
0.8
1

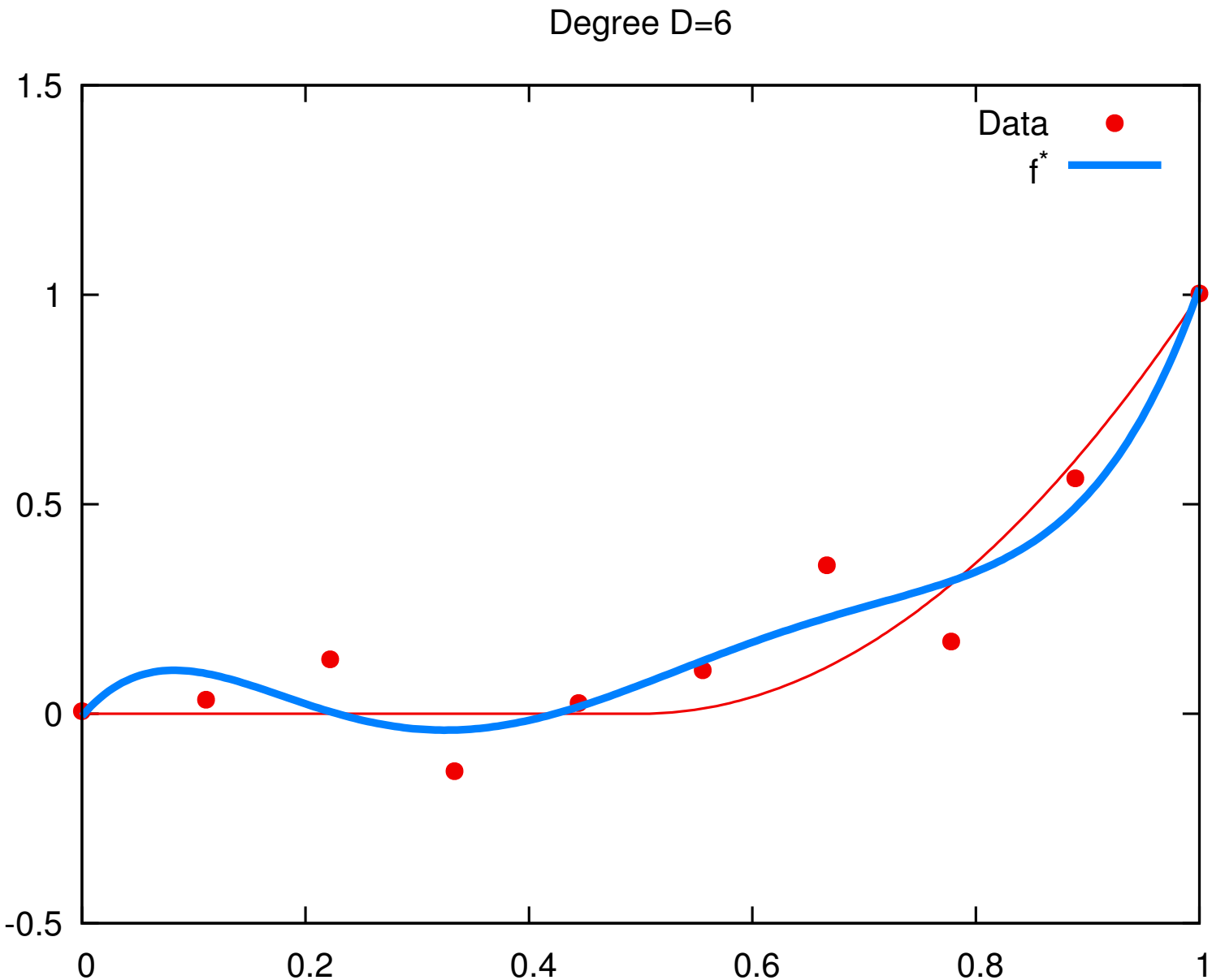

Degree D=1
Data
f*
1.5
1
0.5
0
-0.5
0
0.2
0.4
0.6
0.8
1

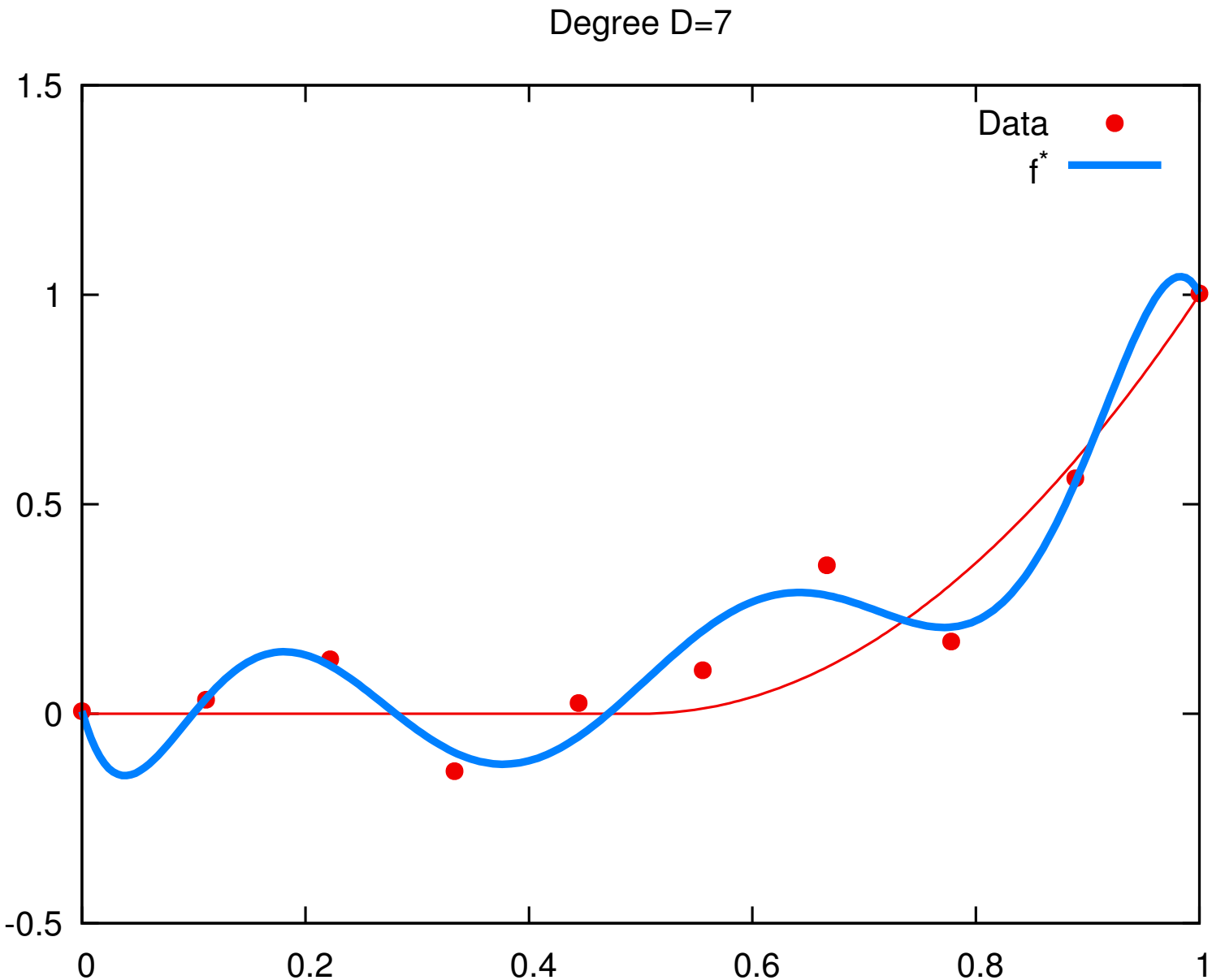

Degree D=2
Data
f*
1.5
1
0.5
0
-0.5
0
0.2
0.4
0.6
0.8
1

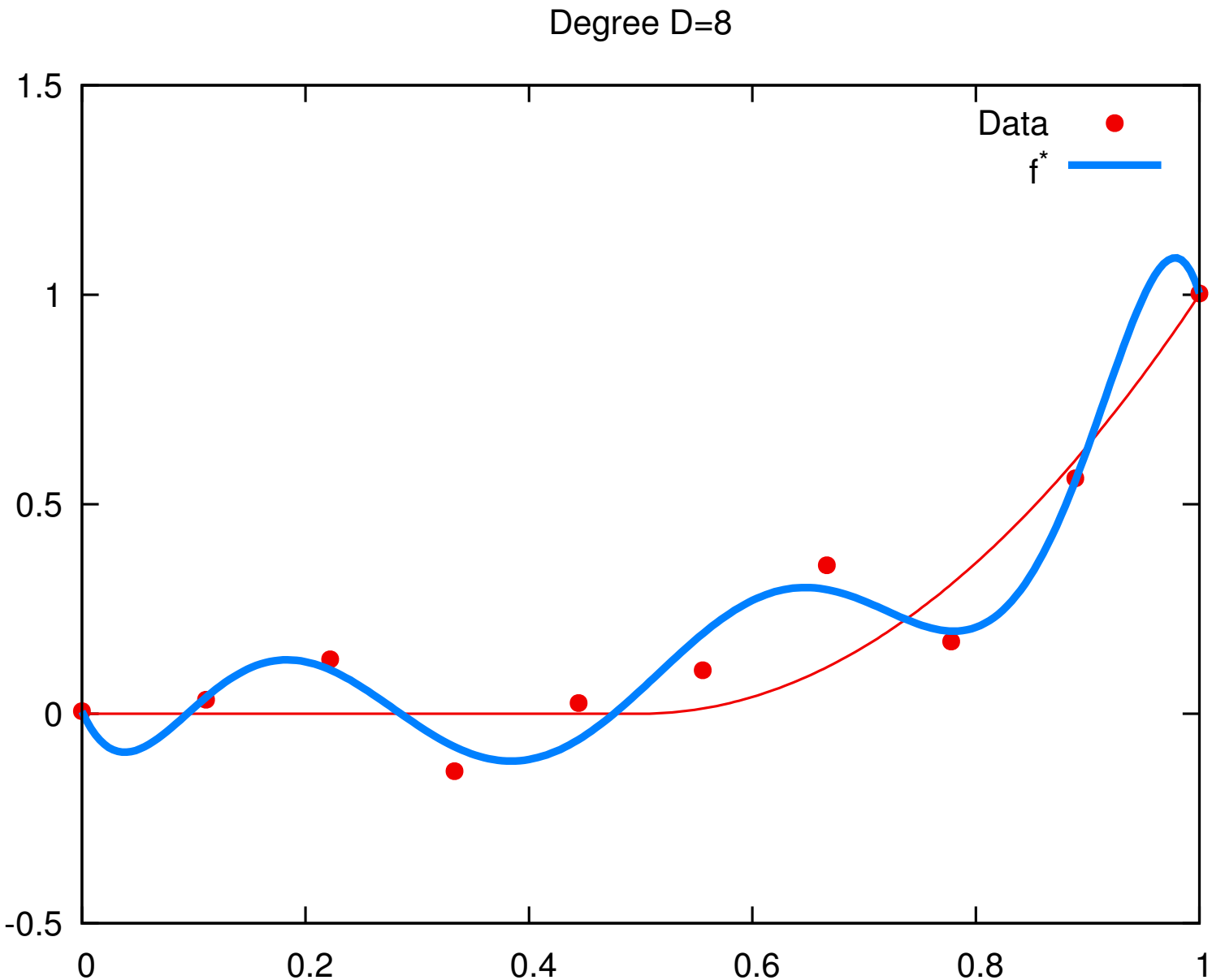

Degree D=3
Data
f*
1.5
1
0.5
0
-0.5
0
0.2
0.4
0.6
0.8
1

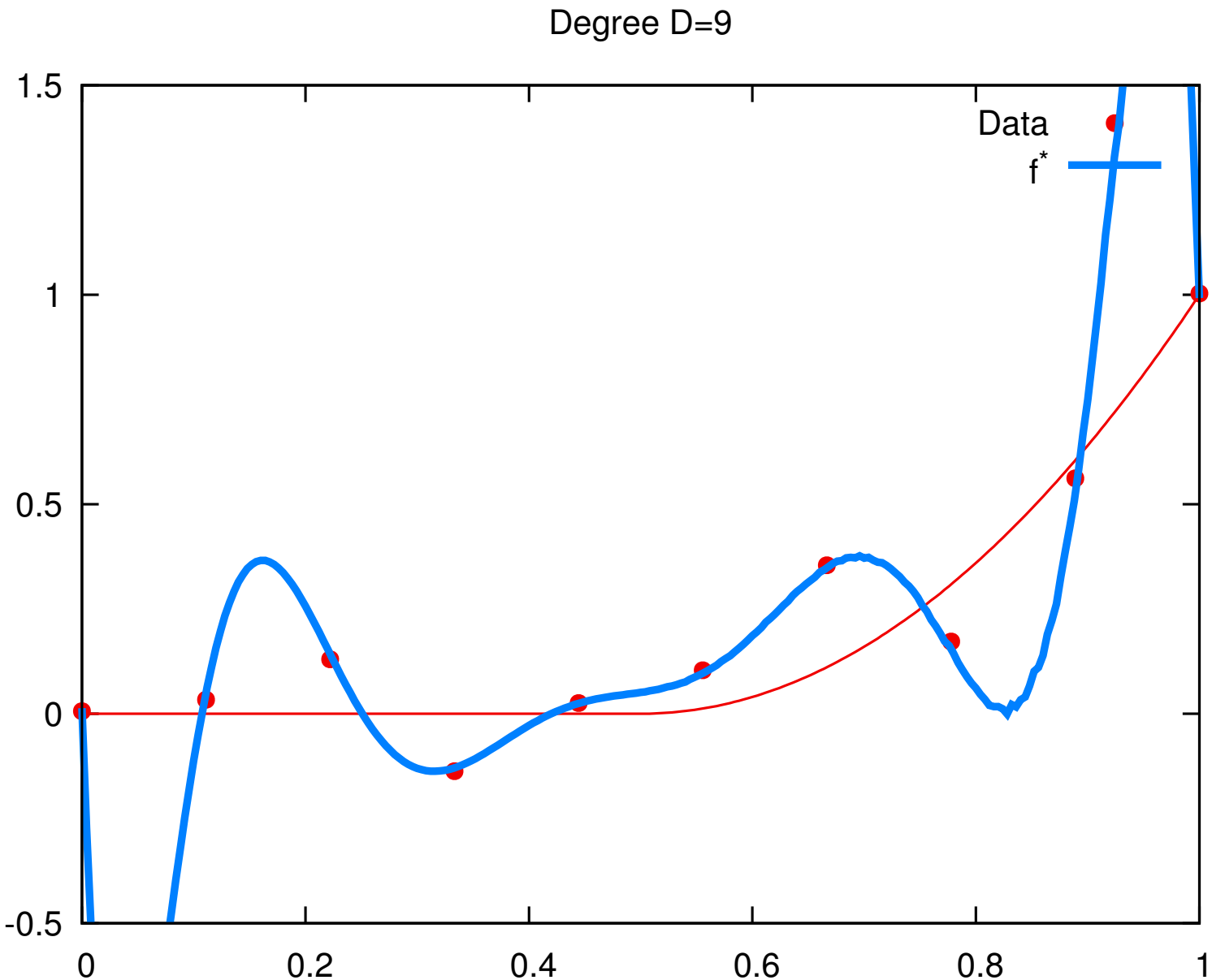

Degree D=4
Data
f*
1.5
1
0.5
0
-0.5
0
0.2
0.4
0.6
0.8
1

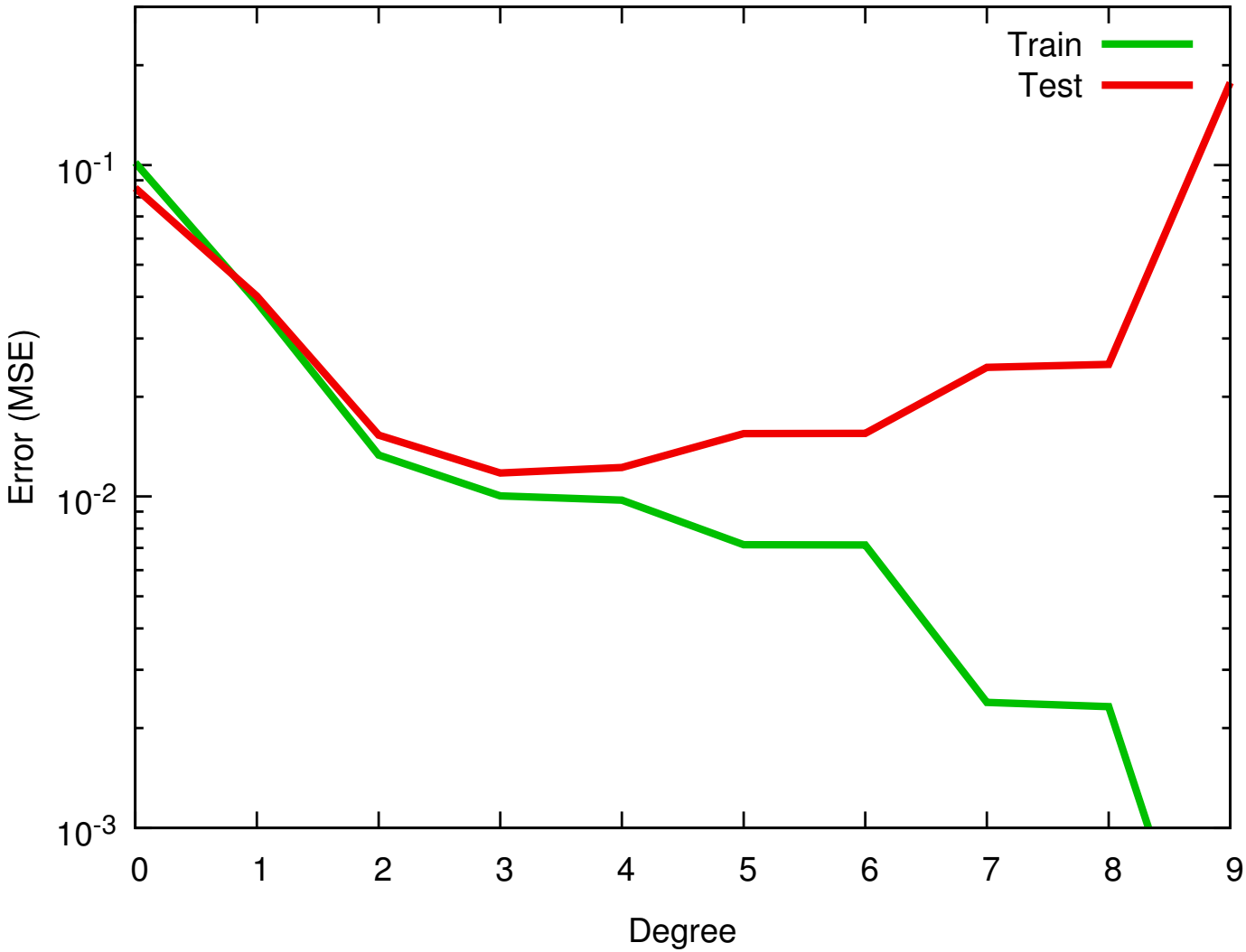

Degree D=5
Data
f*
1.5
1
0.5
0
-0.5
0
0.2
0.4
0.6
0.8
1

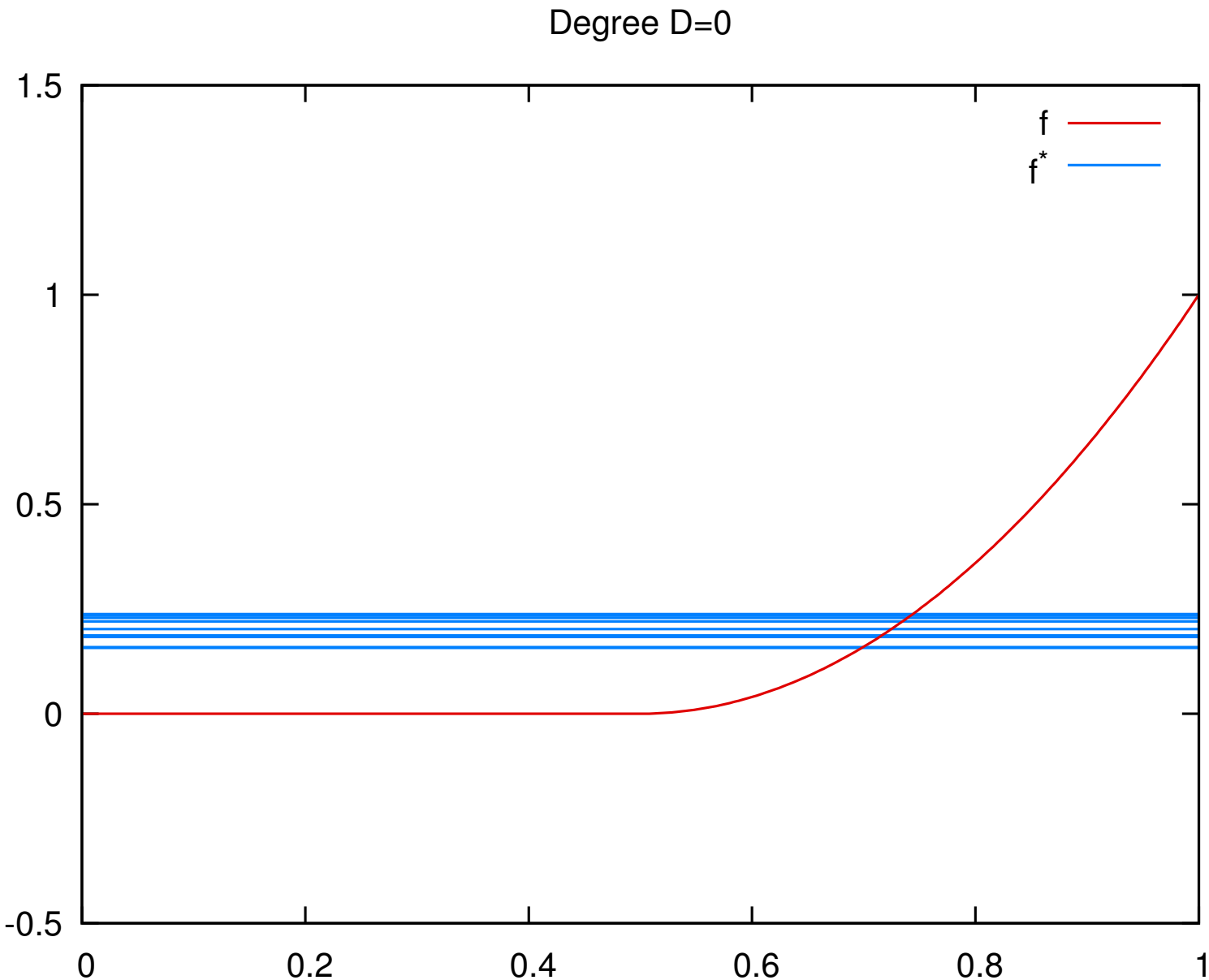

Degree D=6
Data
f*
1.5
1
0.5
0
-0.5
0
0.2
0.4
0.6
0.8
1

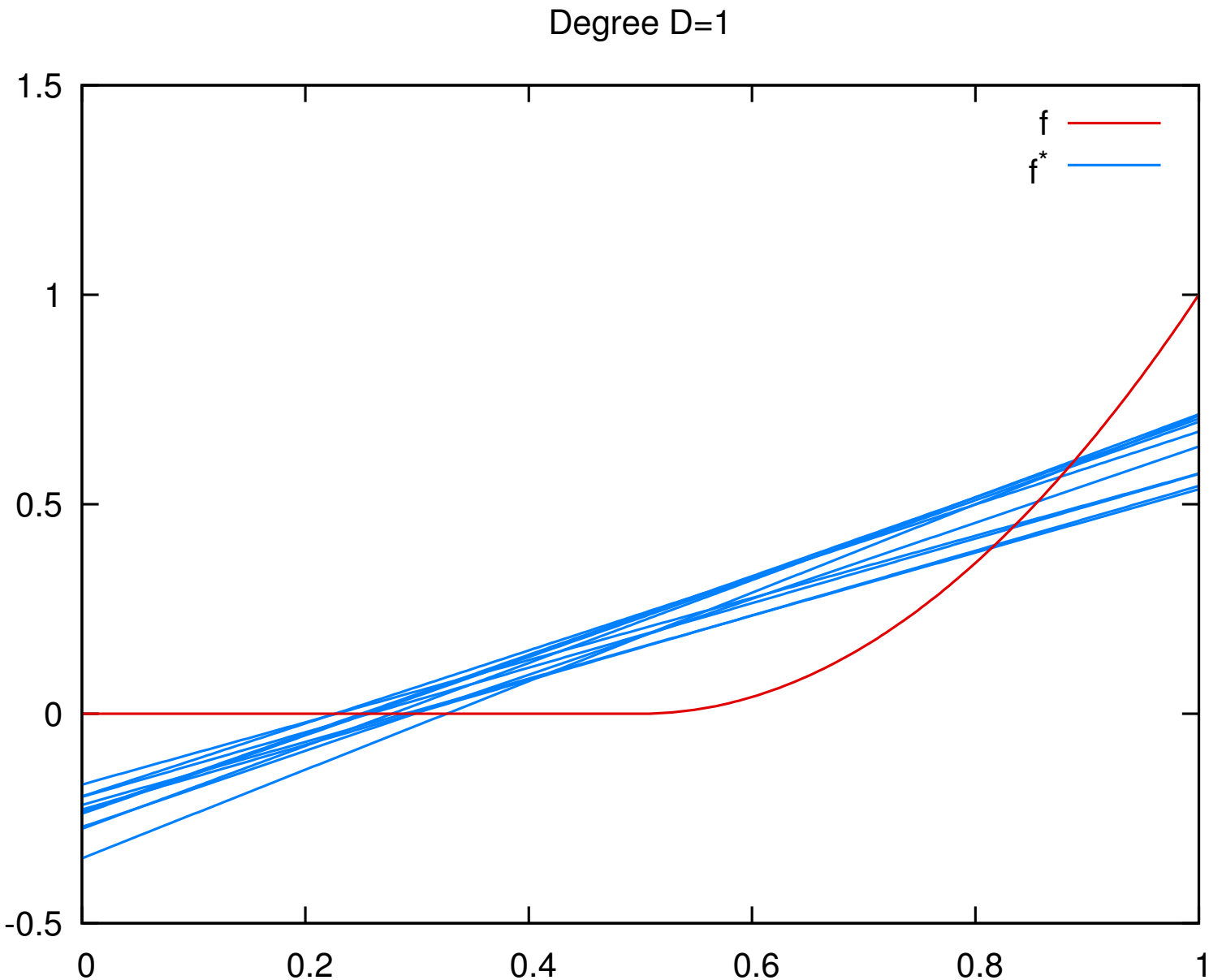

Degree D=7
Data
f*
1.5
1
0.5
0
-0.5
0
0.2
0.4
0.6
0.8
1

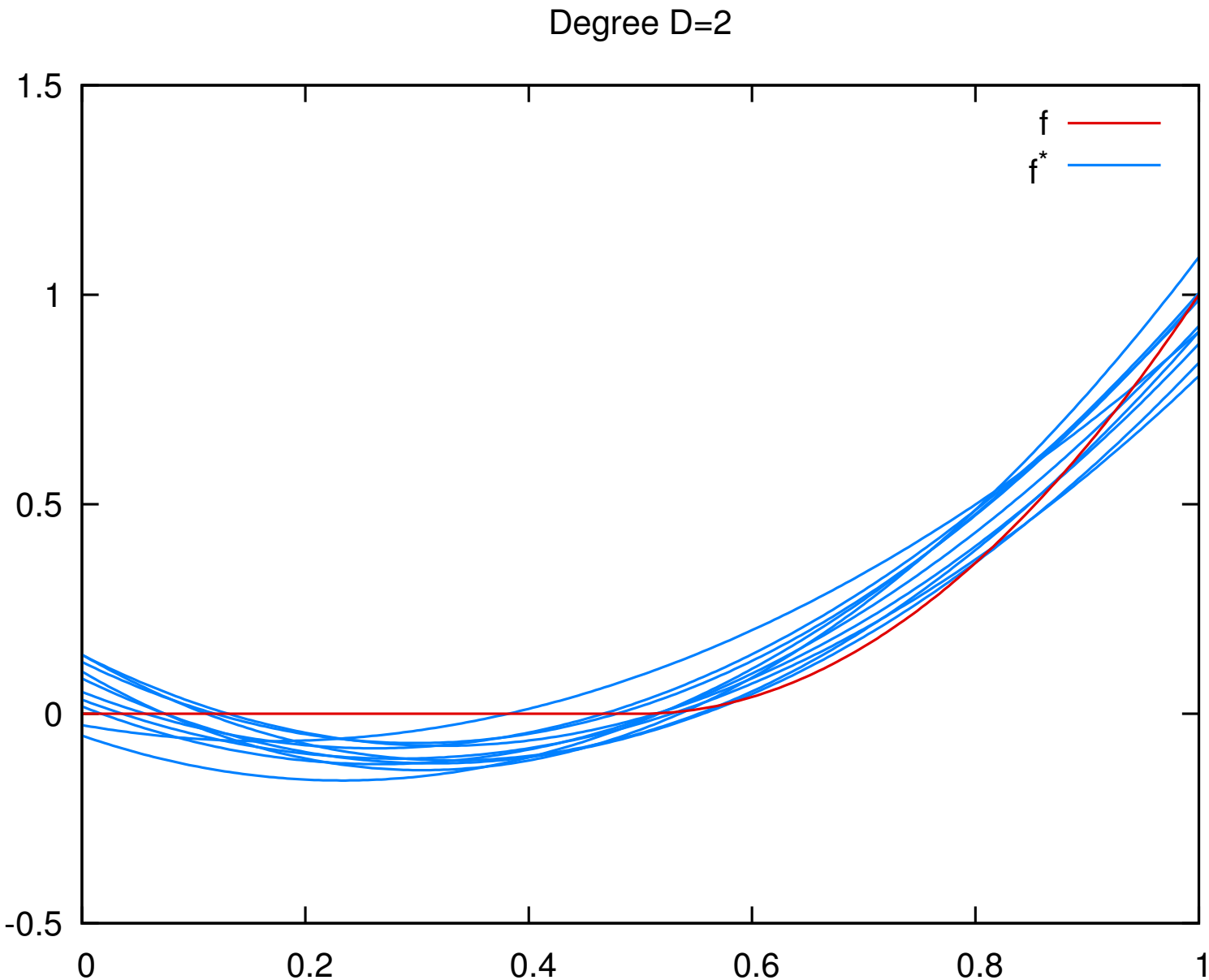

Degree D=8
Data
f*
1.5
1
0.5
0
-0.5
0
0.2
0.4
0.6
0.8
1

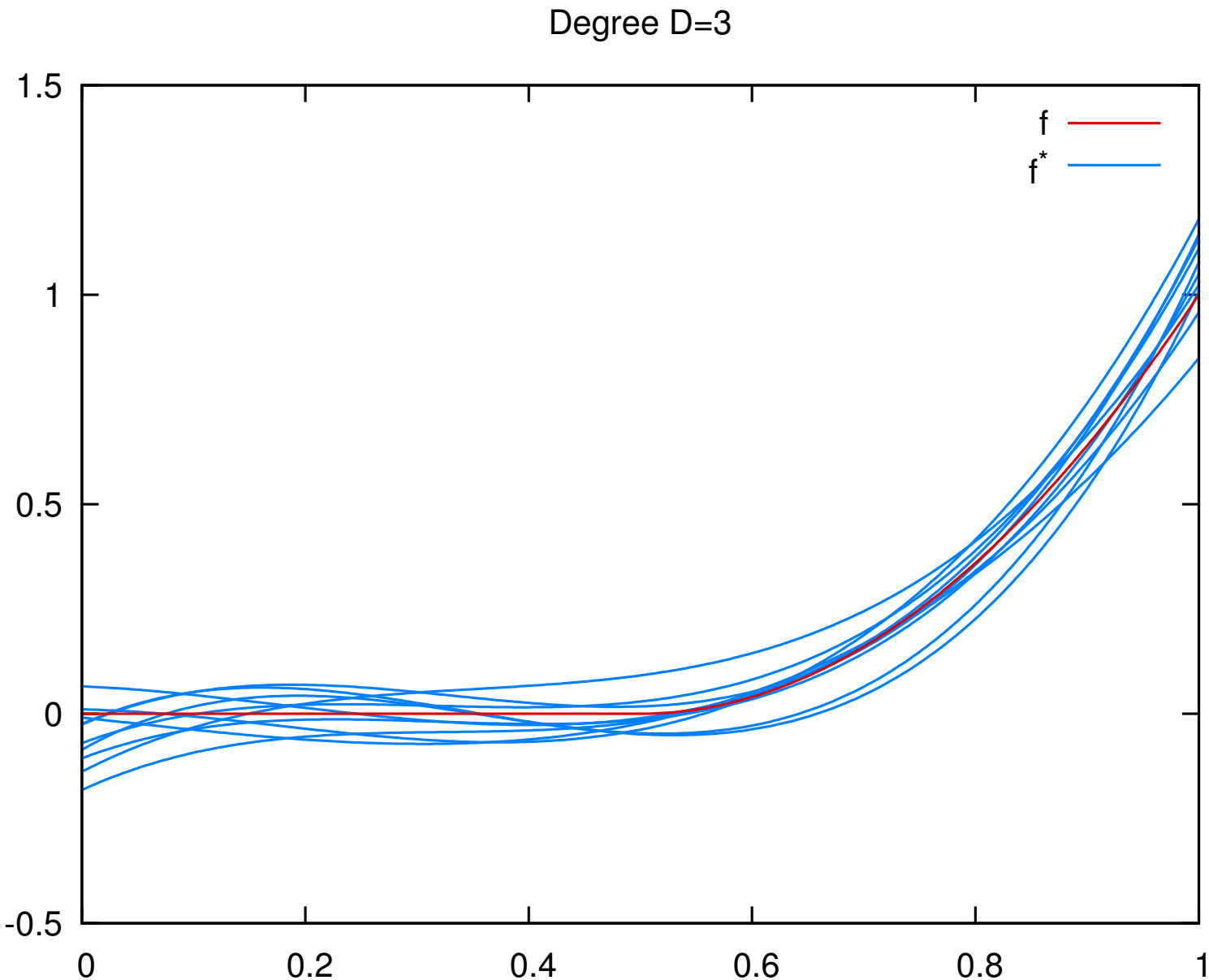

Degree D=9
Data
f*
1.5
1
0.5
0
-0.5
0
0.2
0.4
0.6
0.8
1

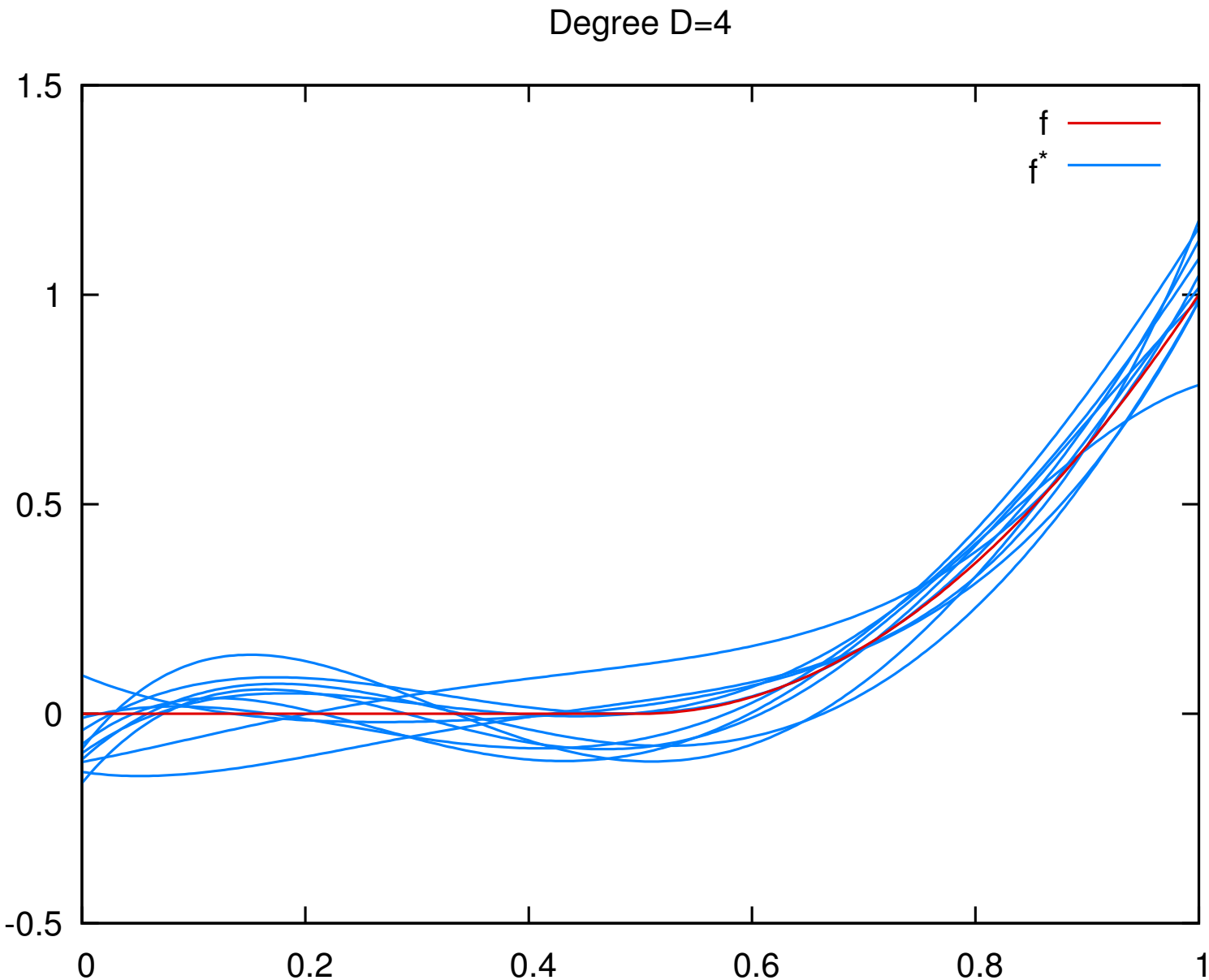

Train
Test
10^{-1}
10^{-2}
10^{-3}
Error (MSE)
0
1
2
3
4
5
6
7
8
9
Degree

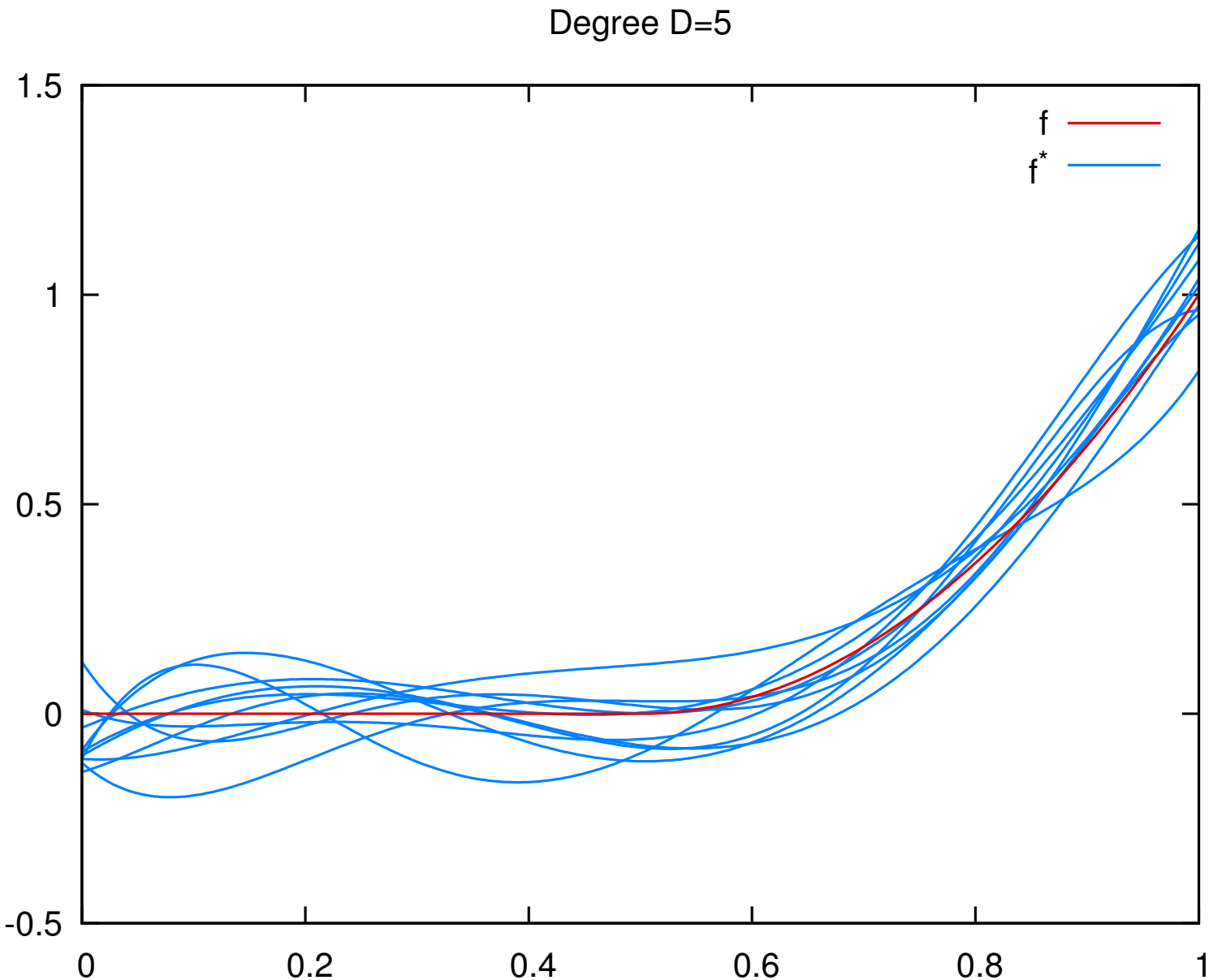

Degree D=0
1.5
1
0.5
0
-0.5
0
0.2
0.4
0.6
0.8
1
f
f*

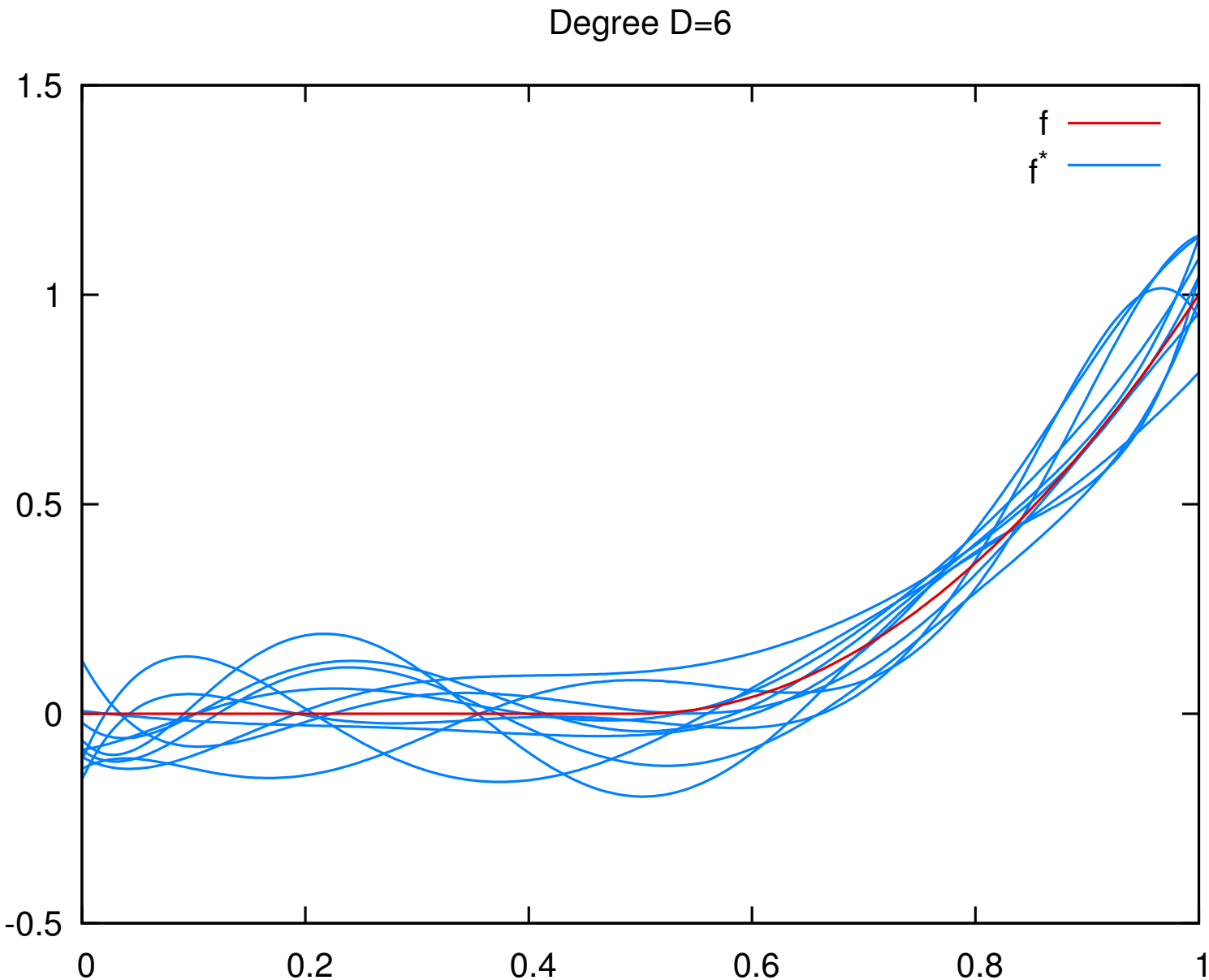

Degree D=1
1.5
1
0.5
0
-0.5
0
0.2
0.4
0.6
0.8
1
f
f*

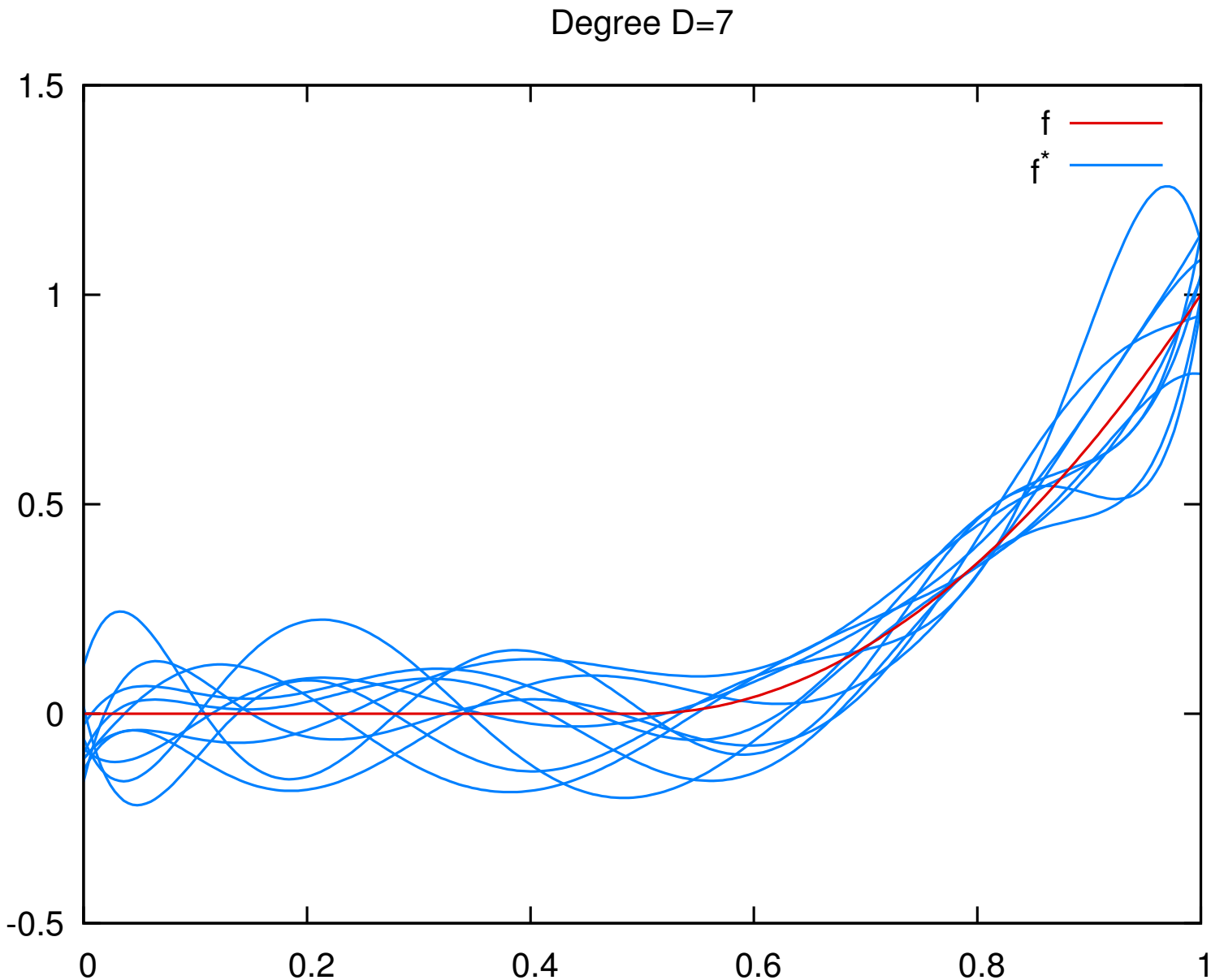

Degree D=2
1.5
1
0.5
0
-0.5
0
0.2
0.4
0.6
0.8
1
f
f*

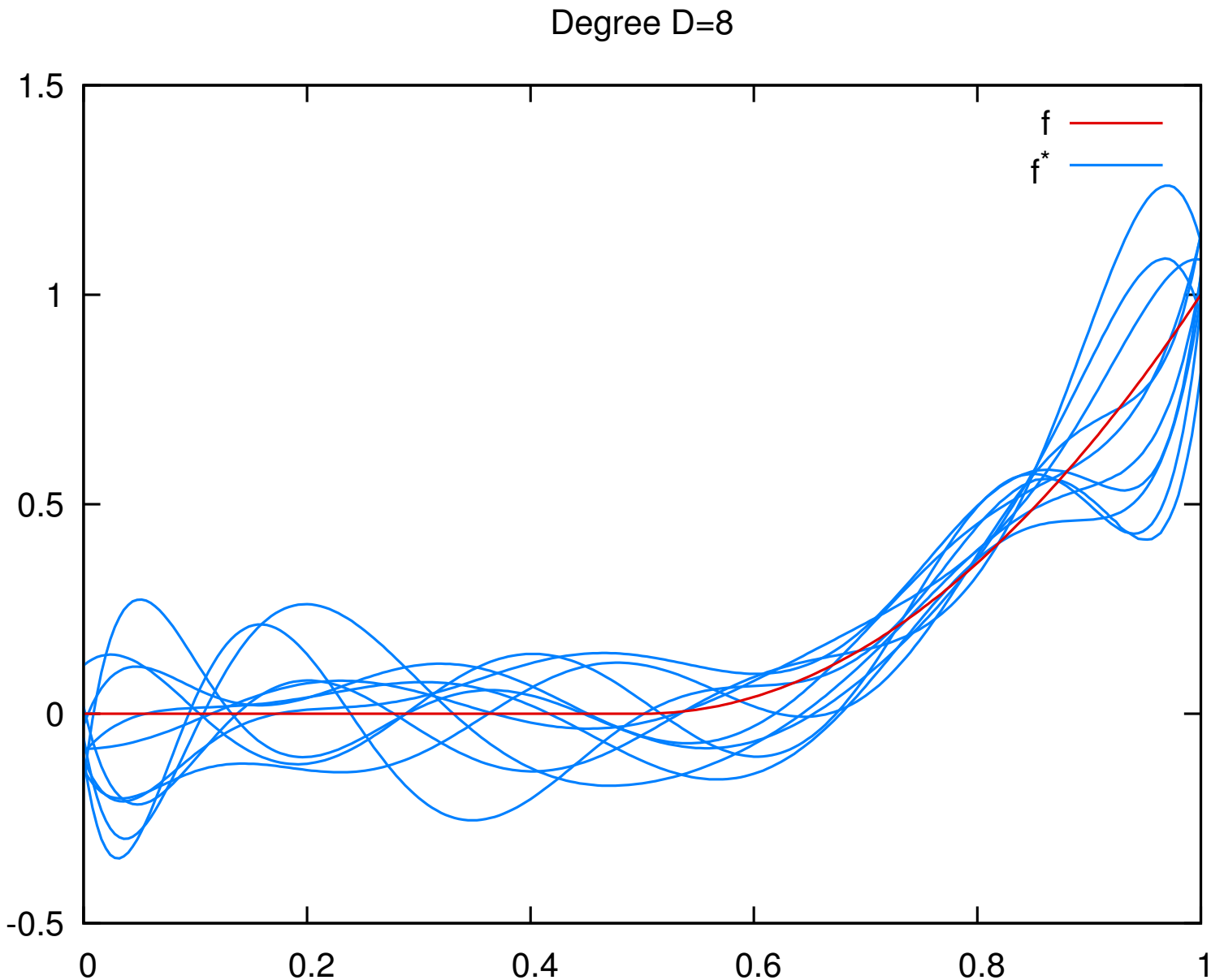

Degree D=3
1.5
1
0.5
0
-0.5
0
0.2
0.4
0.6
0.8
1
f
f*

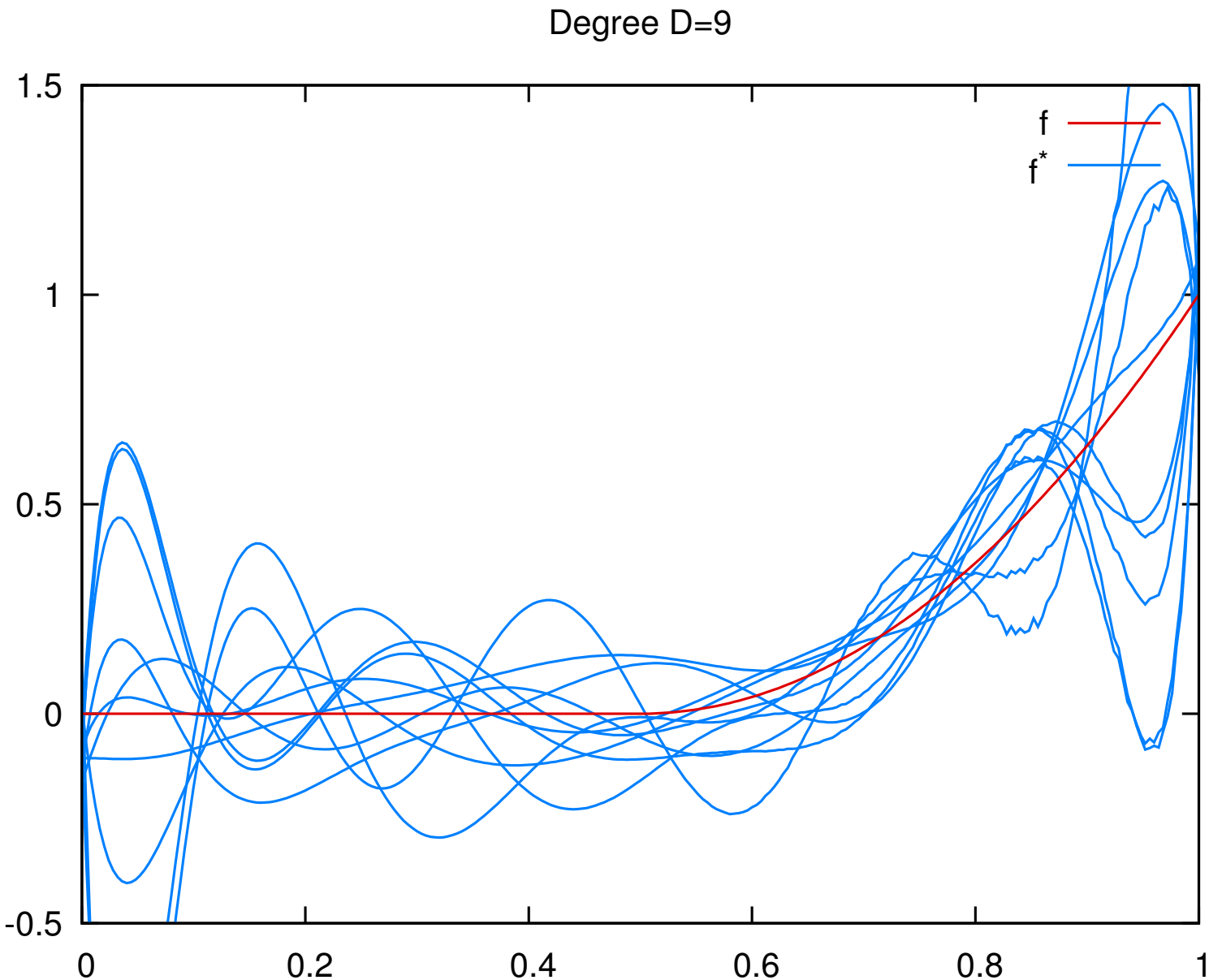

Degree D=4
1.5
1
0.5
0
-0.5
0
0.2
0.4
0.6
0.8
1
f
f*


Degree D=5
f
f*
1.5
1
0.5
0
-0.5
0
0.2
0.4
0.6
0.8
1


Degree D=6
f
f*
1.5
1
0.5
0
-0.5
0
0.2
0.4
0.6
0.8
1


Degree D=7
1.5
1
0.5
0
-0.5
0
0.2
0.4
0.6
0.8
1
f
f*


Degree D=8
1.5
1
0.5
0
-0.5
0
0.2
0.4
0.6
0.8
1
f
f*


Degree D=9
1.5
1
0.5
0
-0.5
0
0.2
0.4
0.6
0.8
1
f
f*

We can reformulate this control of the degree with a penalty

$$\mathscr{L}(\alpha) = \sum_n (f(x_n; \alpha) - y_n)^2 + \sum_d l_d(\alpha_d)$$

where

$$l_d(\alpha) = \begin{cases} 0 & \text{if } d \leq D \text{ or } \alpha = 0 \\ +\infty & \text{otherwise.} \end{cases}$$

Such a penalty kills any term of degree $> D$.

We can reformulate this control of the degree with a penalty

$$\mathscr{L}(\alpha) = \sum_n (f(x_n;\alpha) - y_n)^2 + \sum_d l_d(\alpha_d)$$

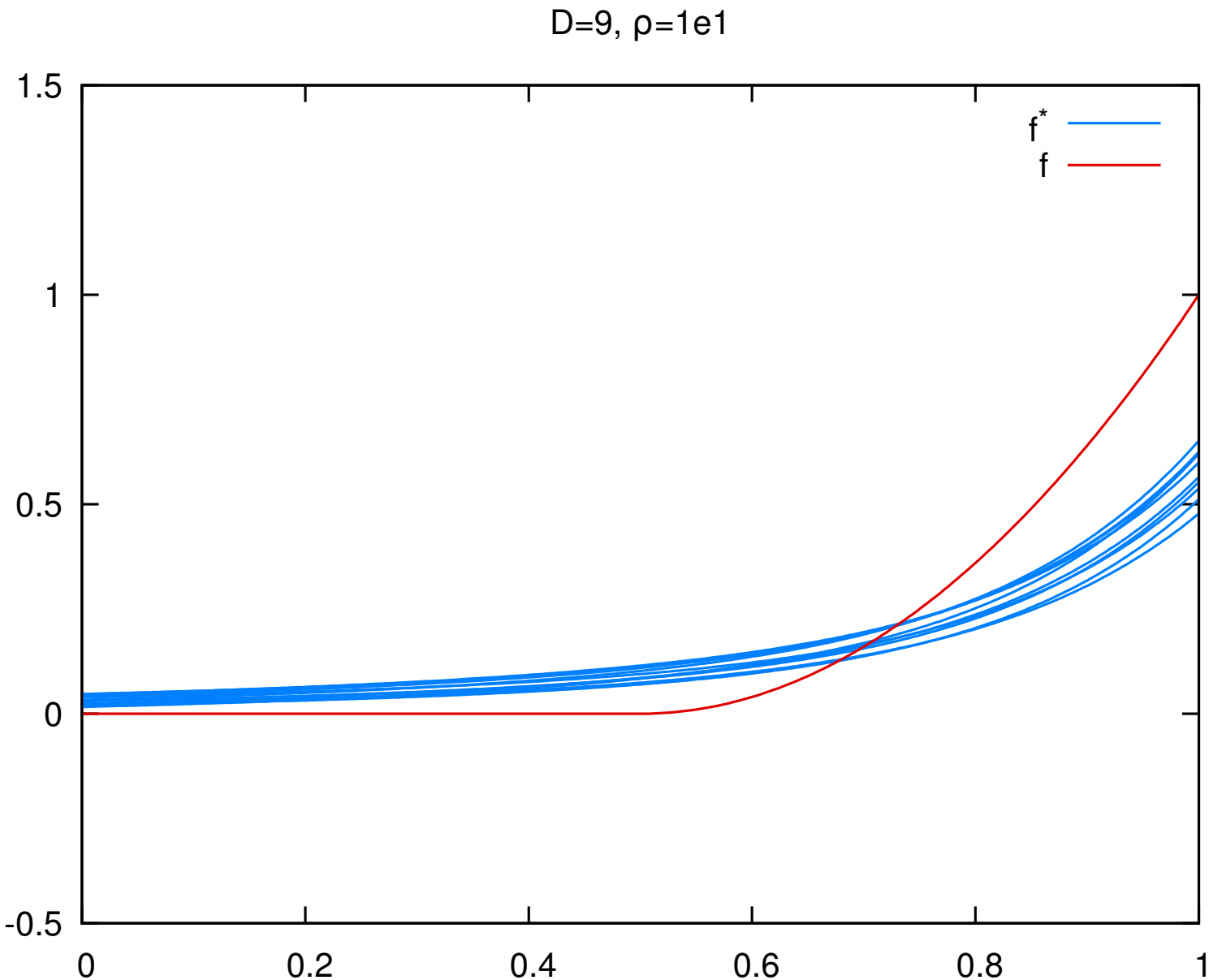
where

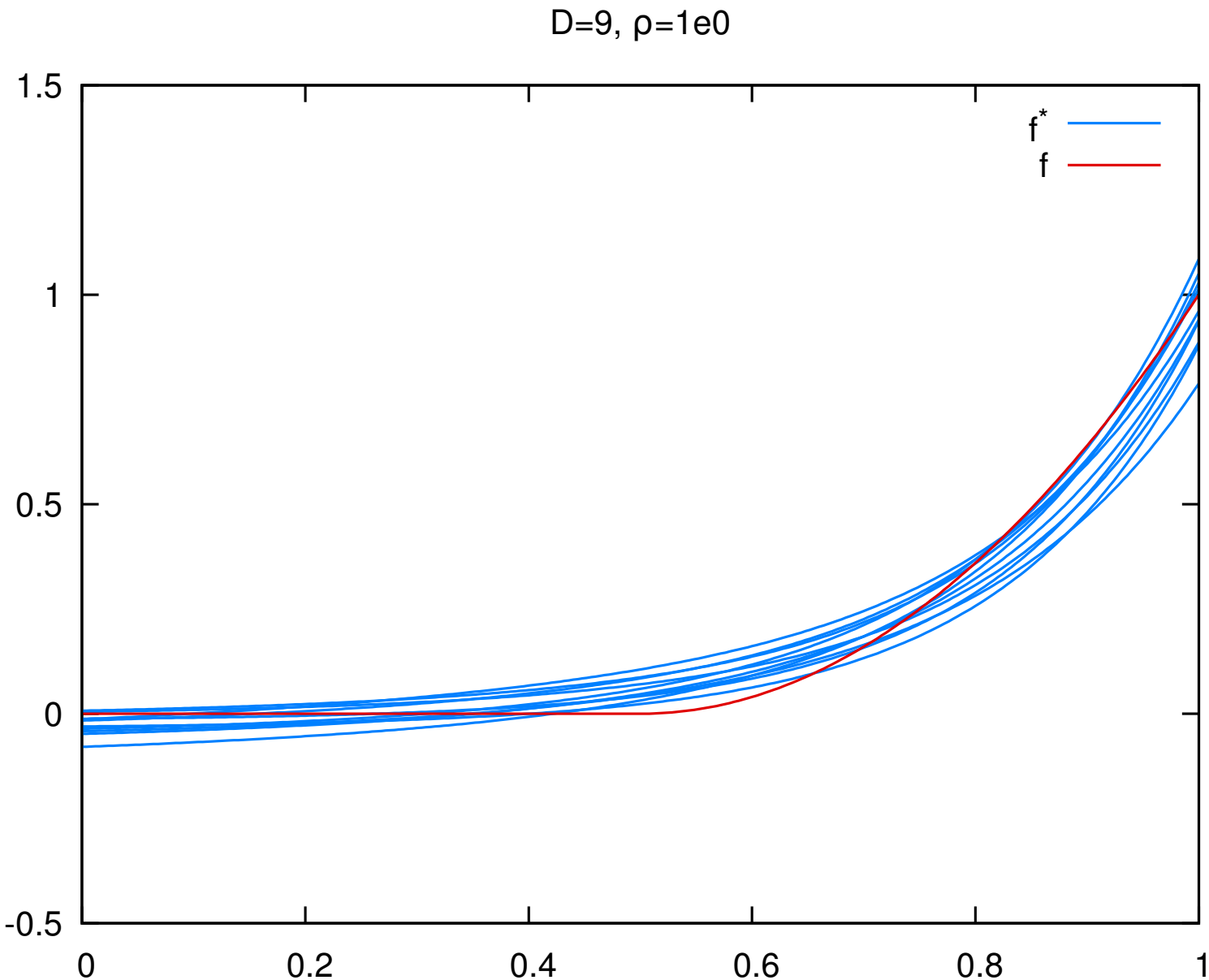
$$l_d(\alpha) = \begin{cases} 0 & \text{if } d \leq D \text{ or } \alpha = 0 \\ +\infty & \text{otherwise.} \end{cases}$$

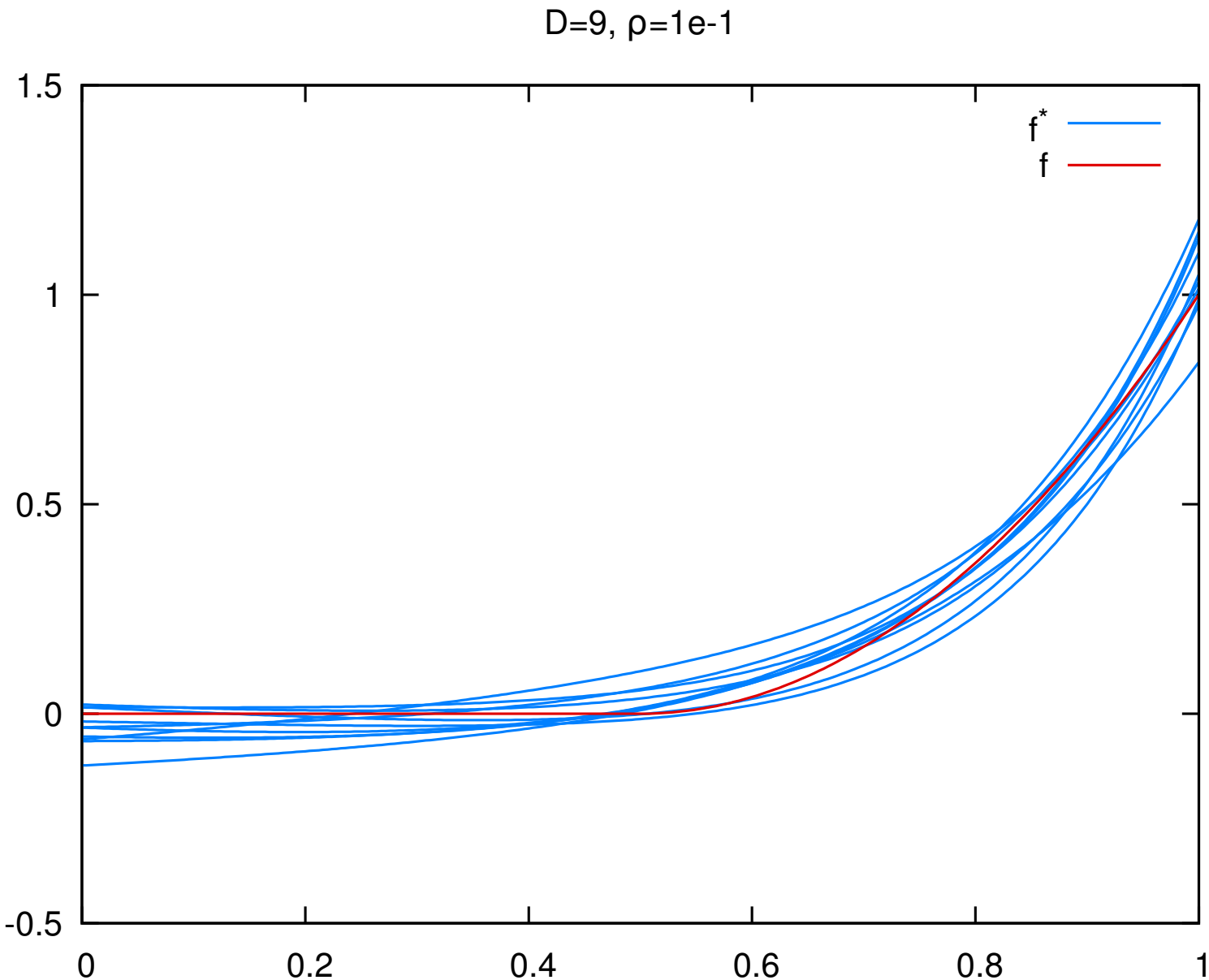
Such a penalty kills any term of degree $> D$.

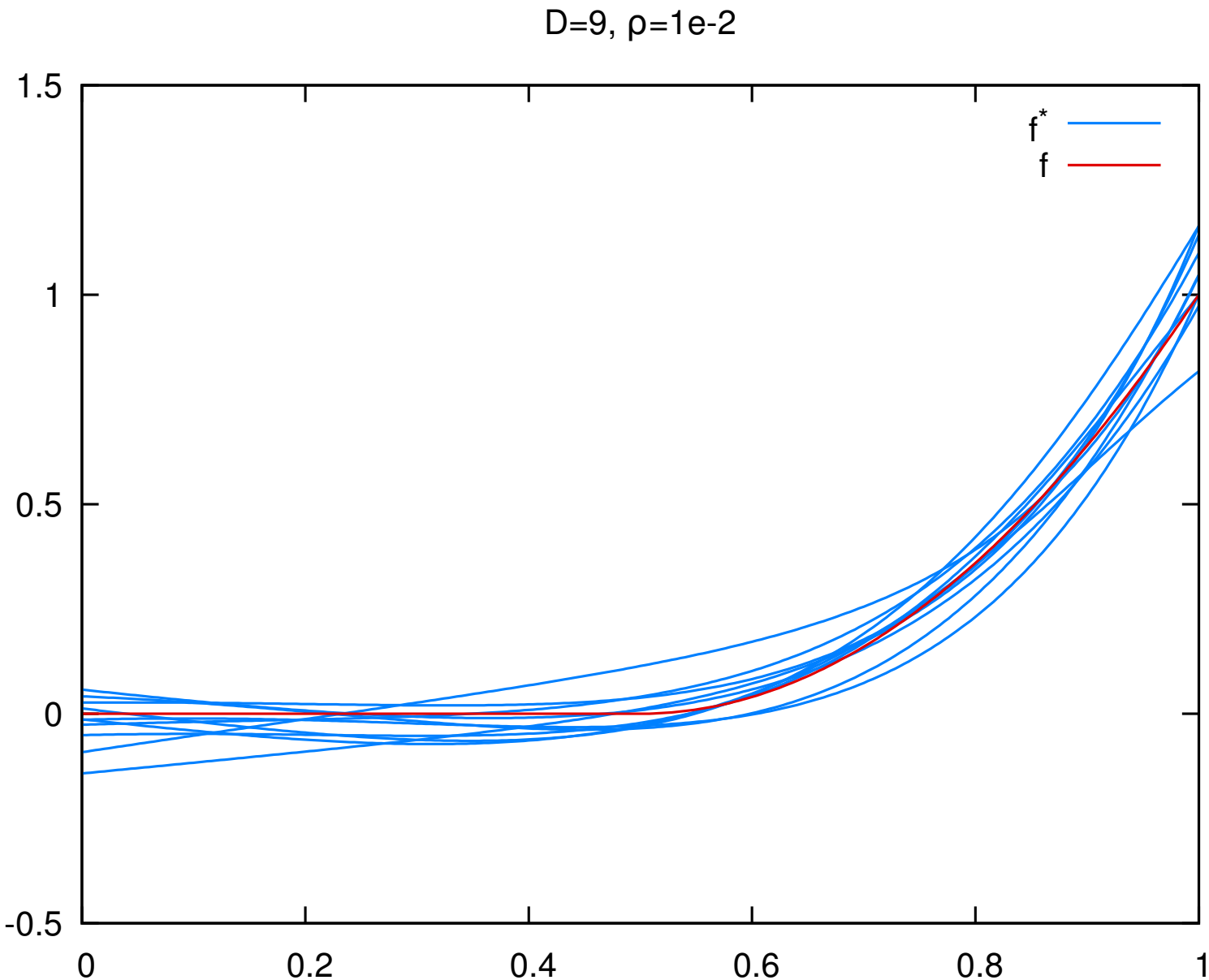
This motivates the use of more subtle variants. For instance, to keep all this quadratic

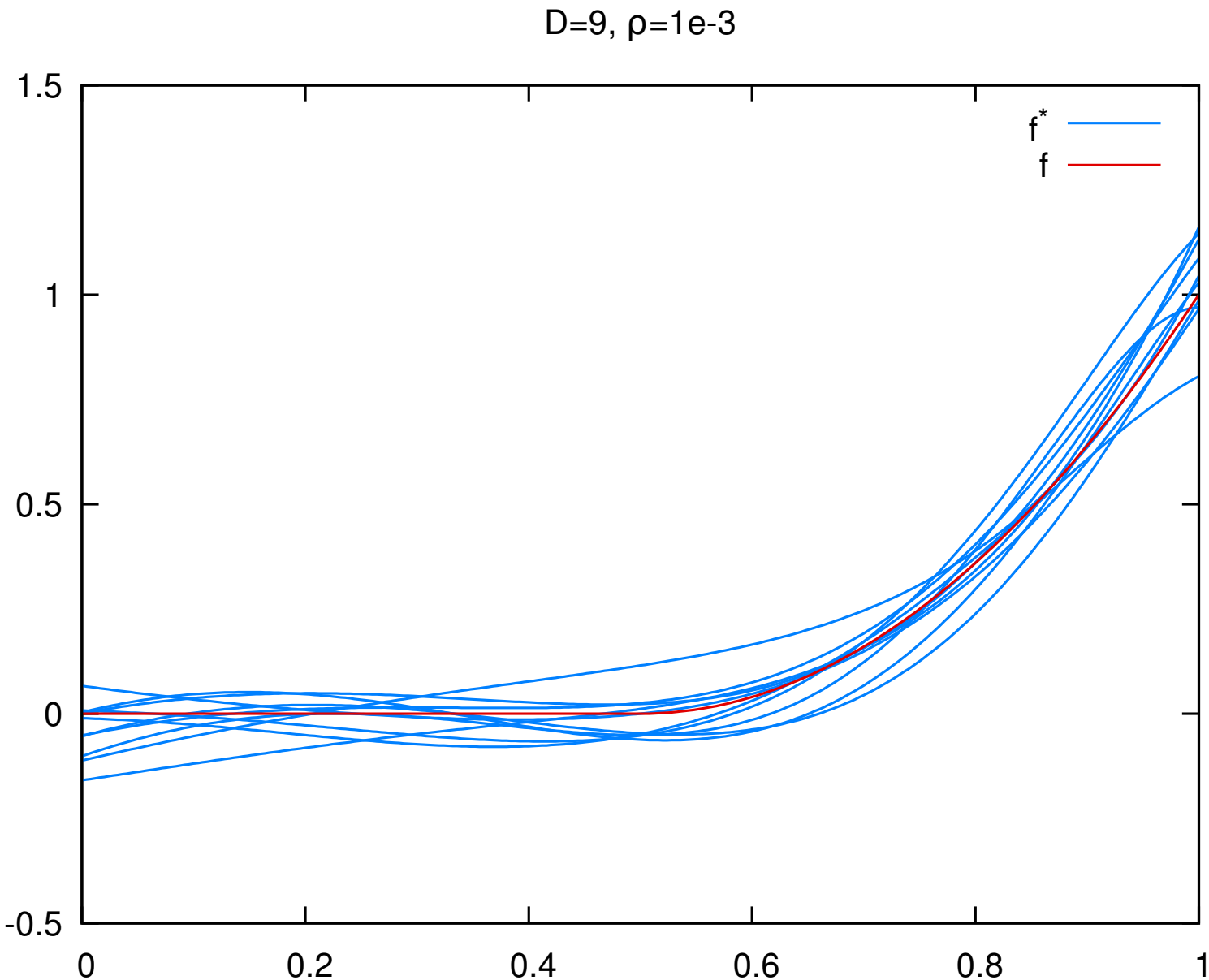
$$\mathscr{L}(\alpha) = \sum_n (f(x_n;\alpha) - y_n)^2 + \rho \sum_d \alpha_d^2.$$

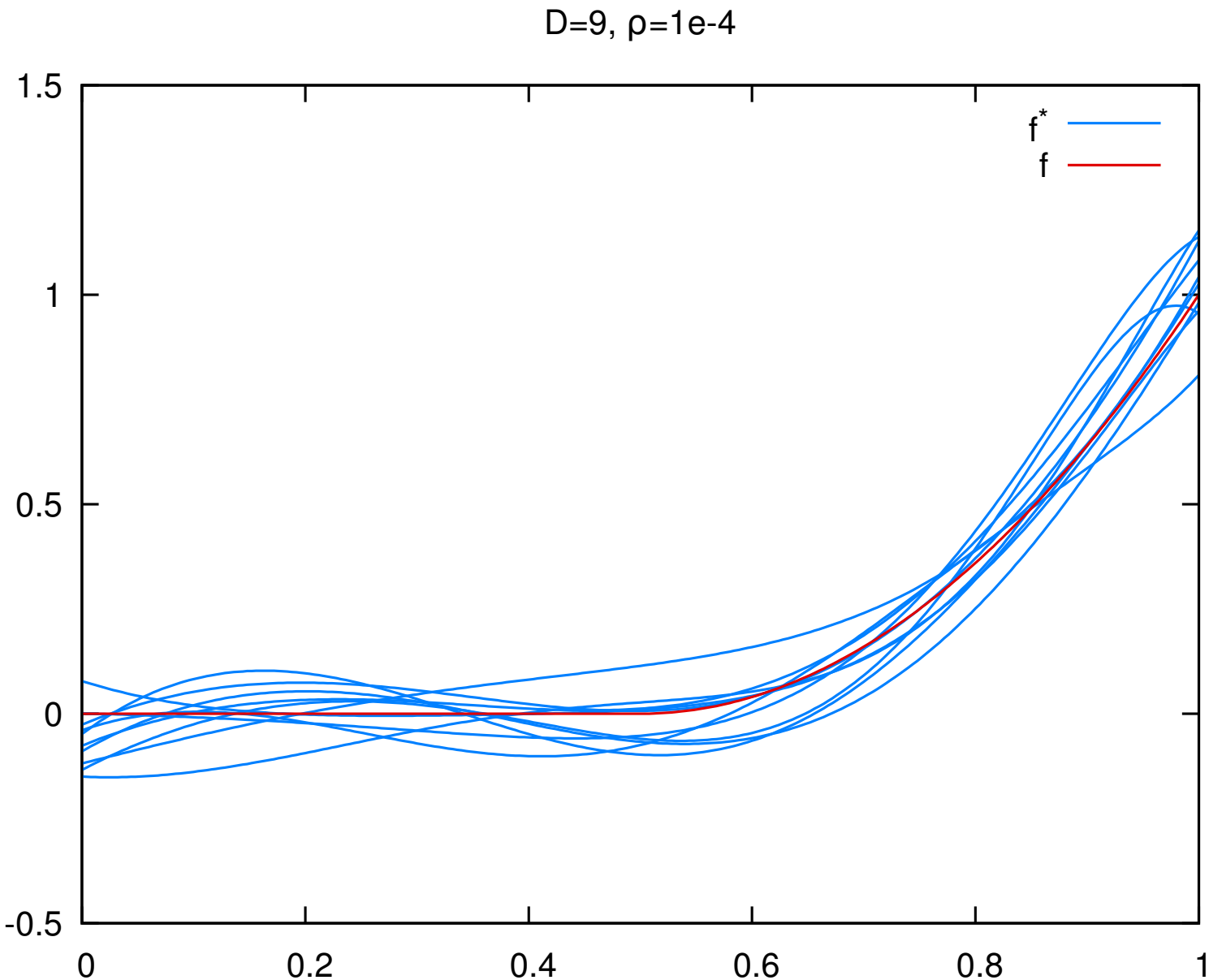

D=9, ρ=1e1
1.5
1
0.5
0
-0.5
0
0.2
0.4
0.6
0.8
1
f*
f

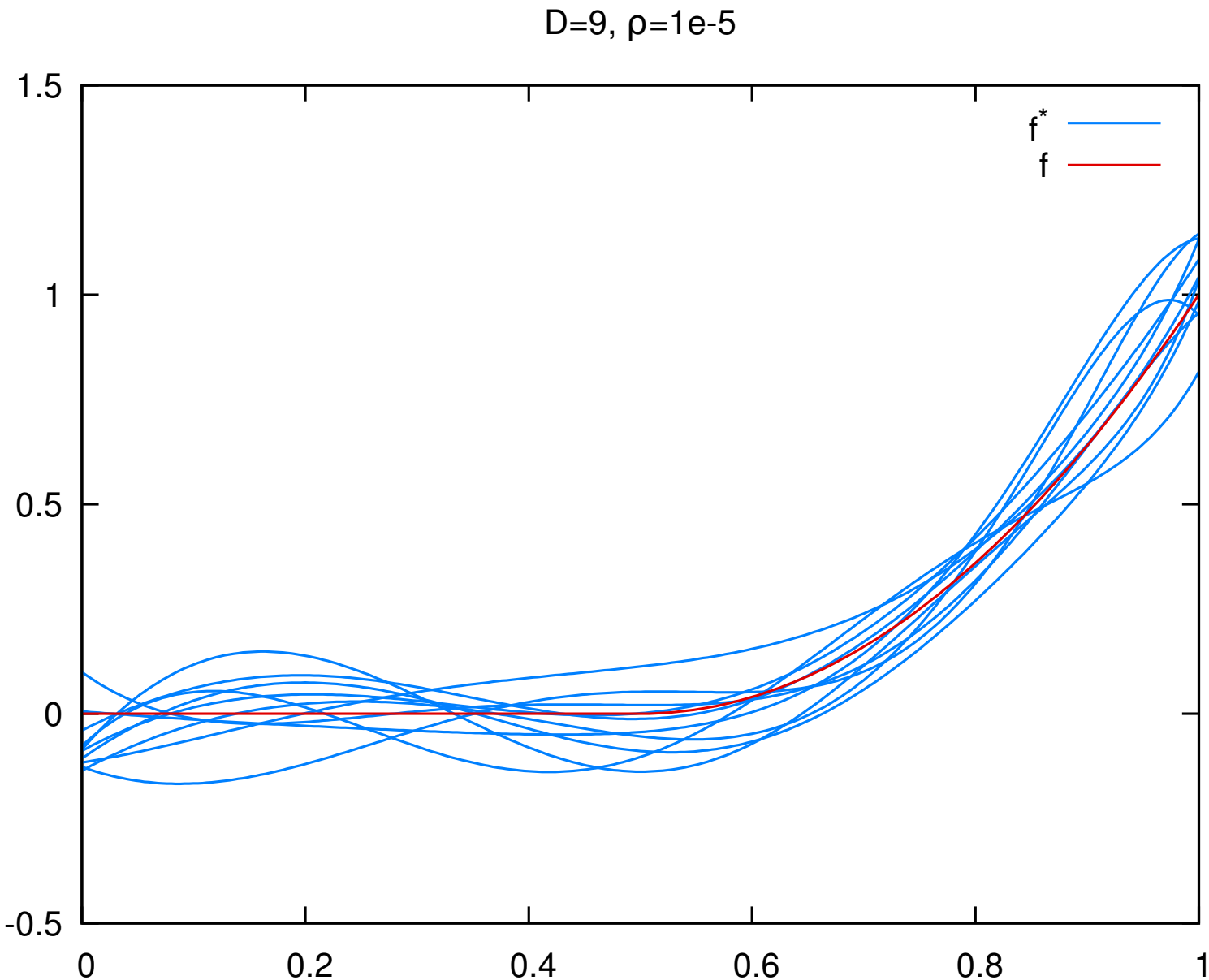

D=9, ρ=1e0
1.5
1
0.5
0
-0.5
0
0.2
0.4
0.6
0.8
1
f*
f

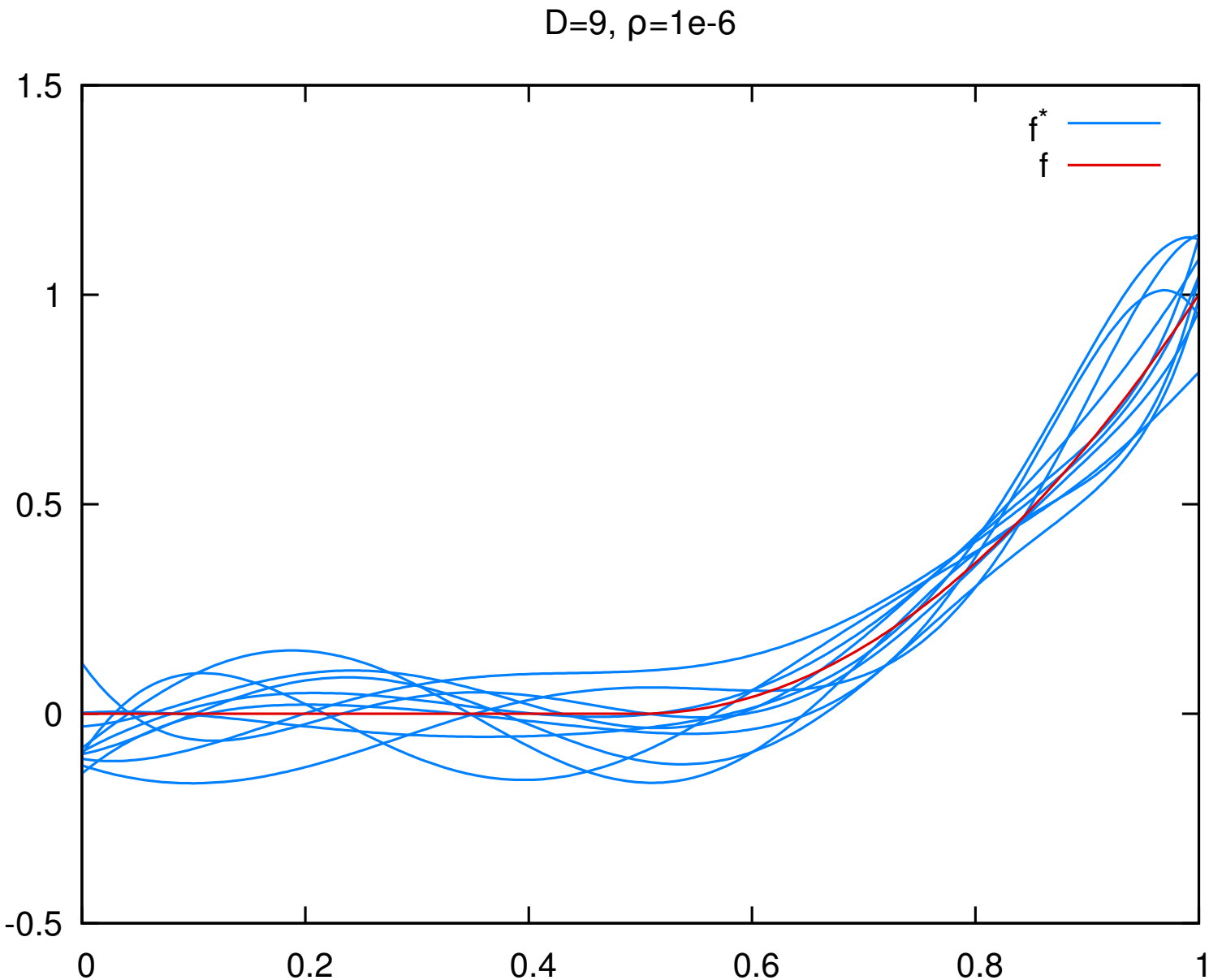

D=9, ρ=1e-1
1.5
1
0.5
0
-0.5
0
0.2
0.4
0.6
0.8
1
f*
f

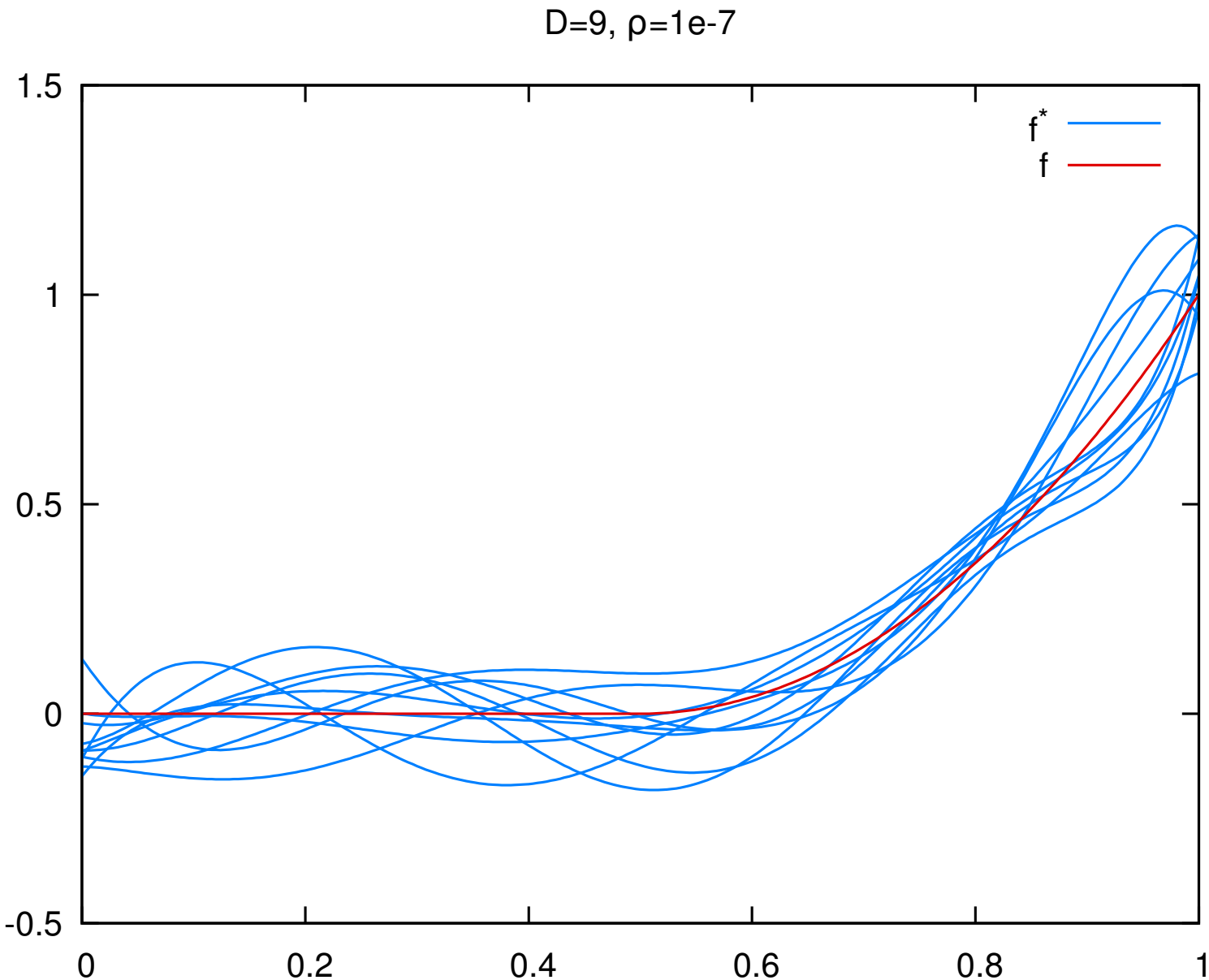

D=9, ρ=1e-2
1.5
1
0.5
0
-0.5
0
0.2
0.4
0.6
0.8
1
f*
f

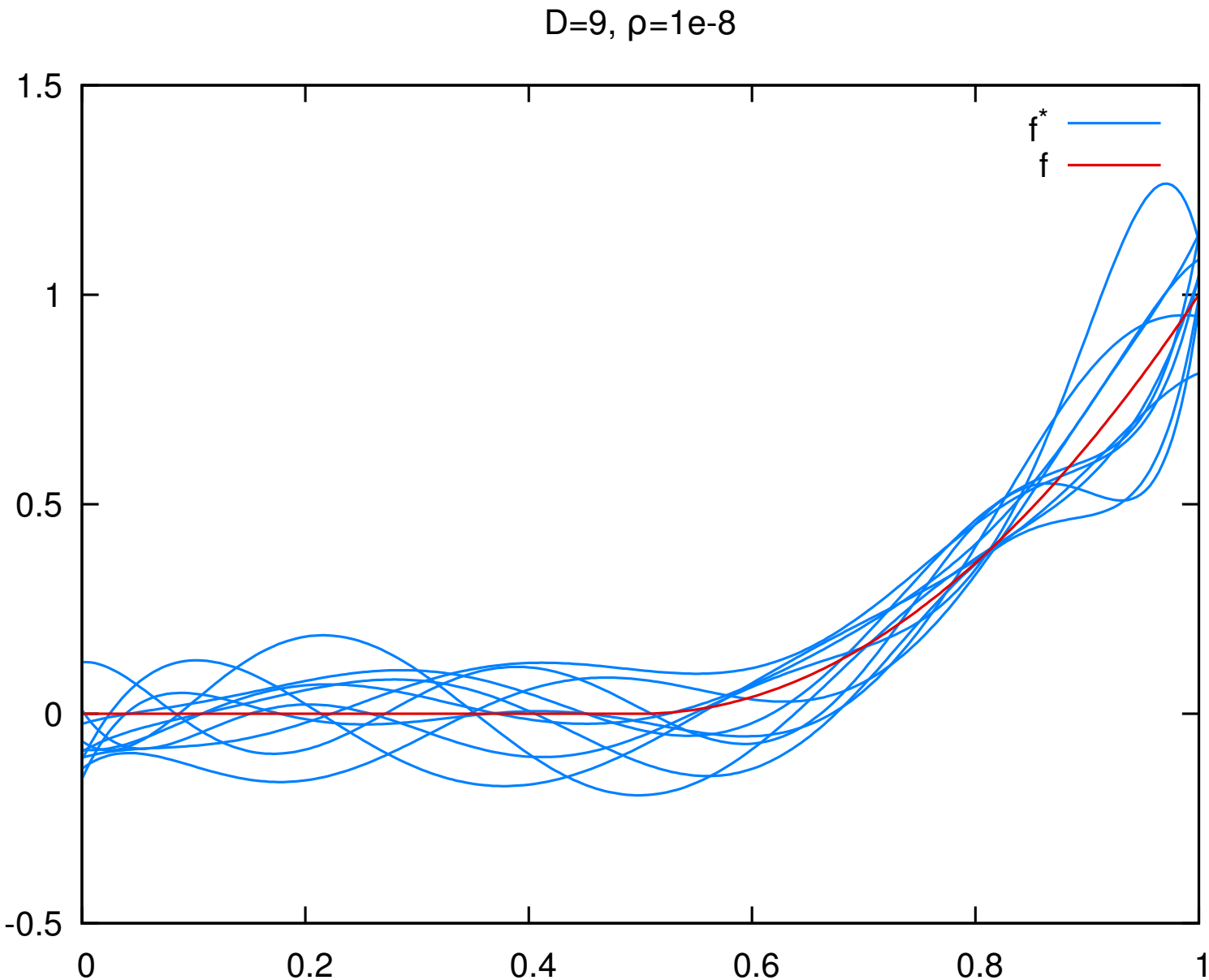

D=9, ρ=1e-3
1.5
1
0.5
0
-0.5
0
0.2
0.4
0.6
0.8
1
f*
f

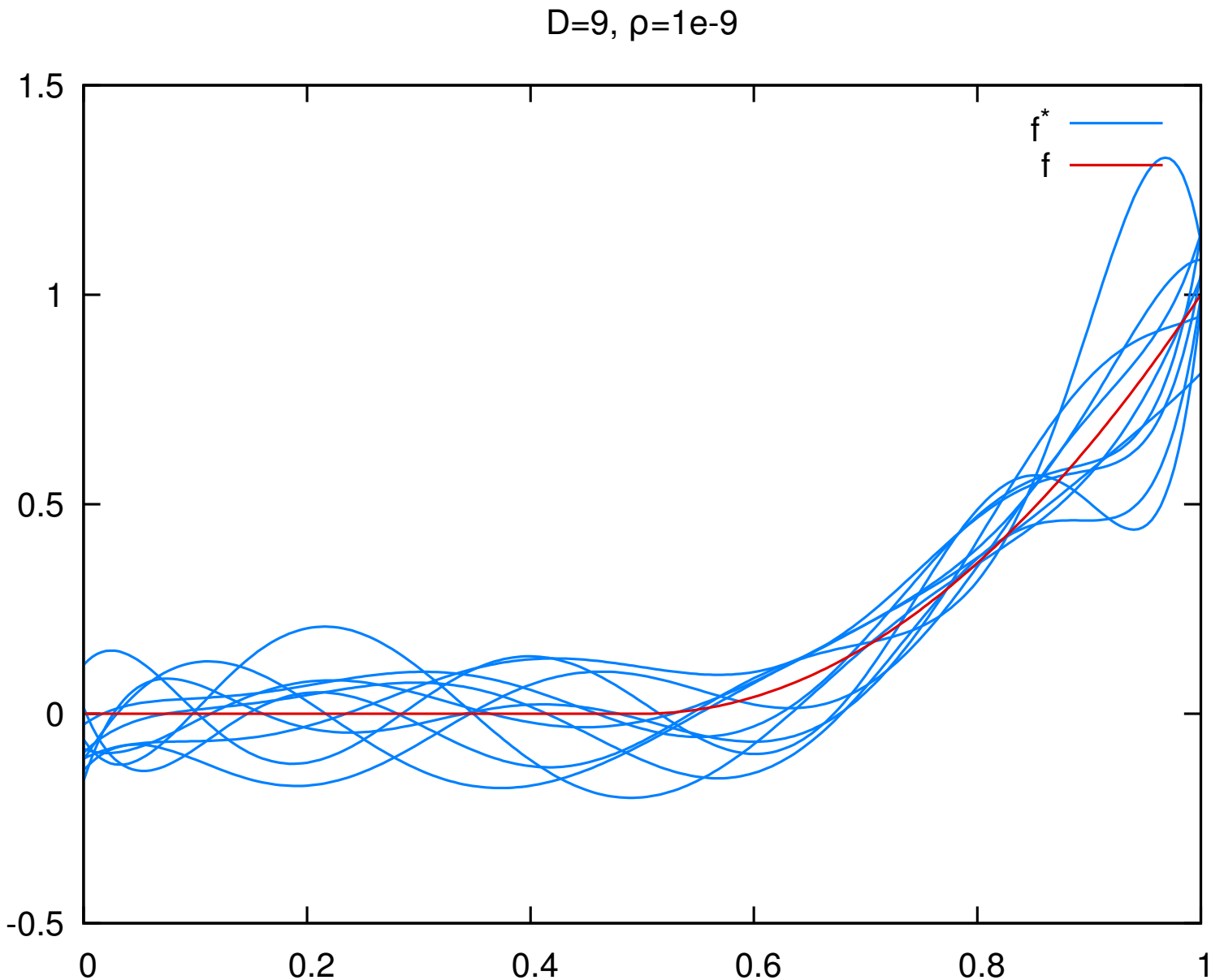

D=9, ρ=1e-4
1.5
1
0.5
0
-0.5
0
0.2
0.4
0.6
0.8
1
f*
f

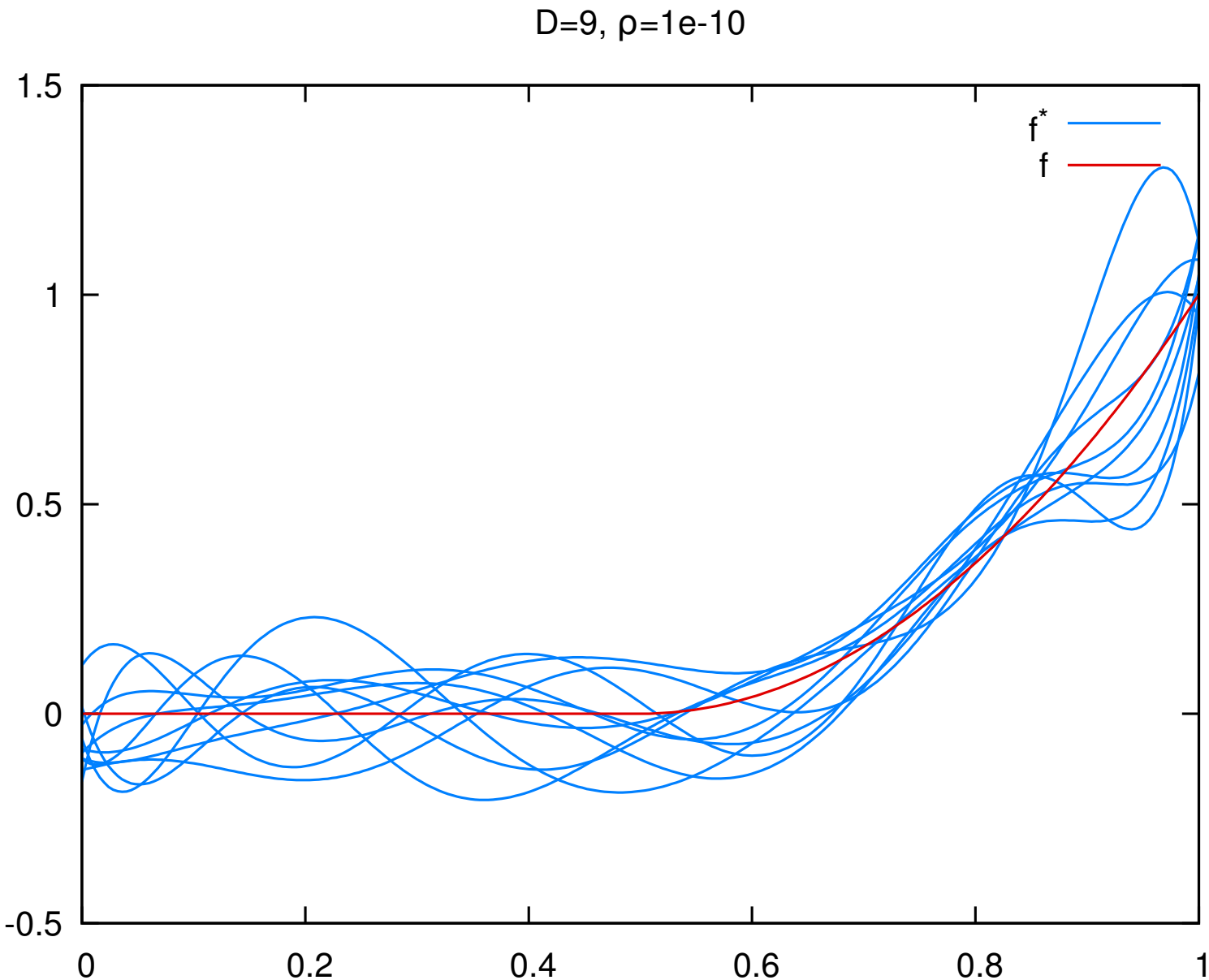

D=9, ρ=1e-5
1.5
1
0.5
0
-0.5
0
0.2
0.4
0.6
0.8
1
f*
f

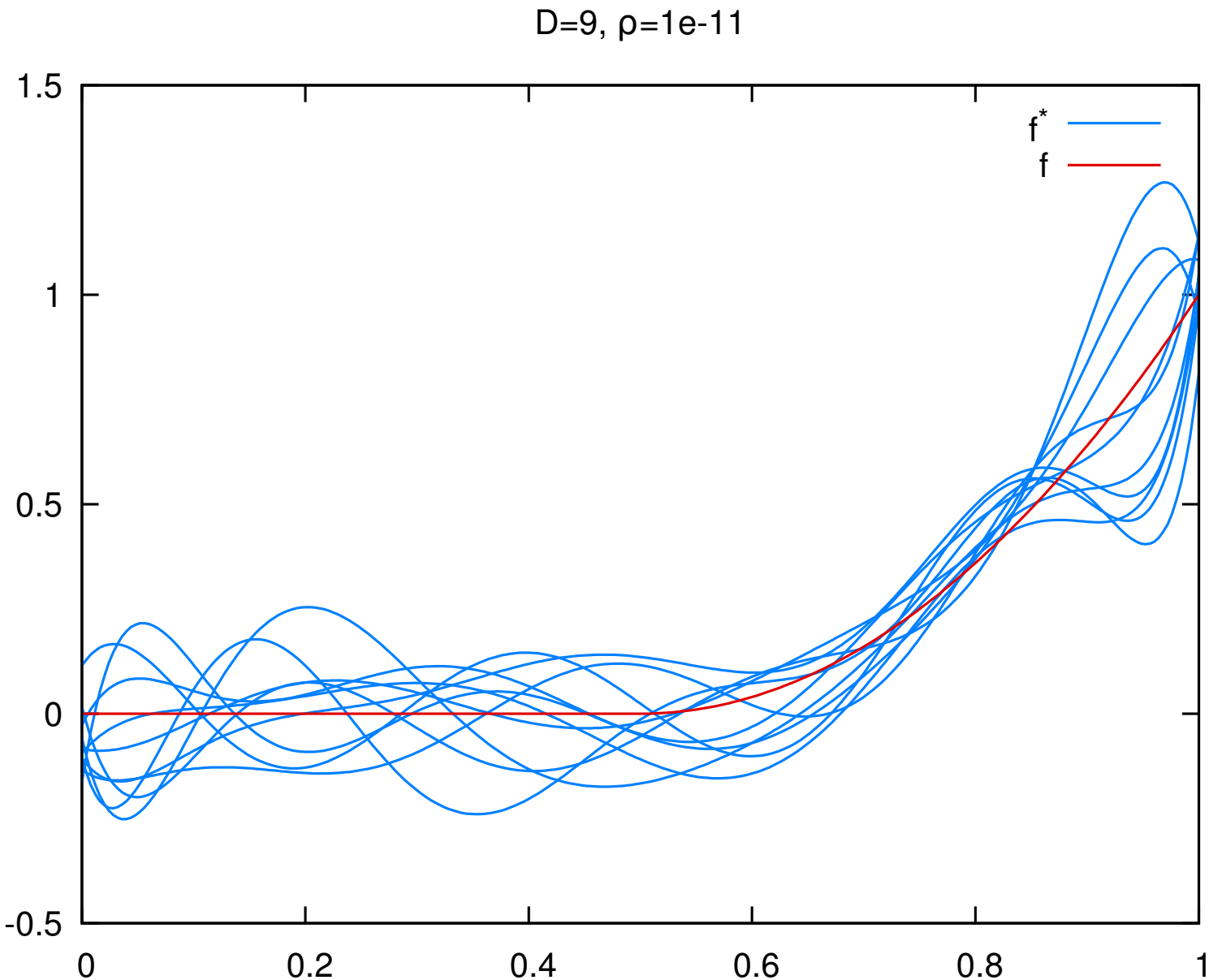

D=9, ρ=1e-6
1.5
1
0.5
0
-0.5
0
0.2
0.4
0.6
0.8
1
f*
f

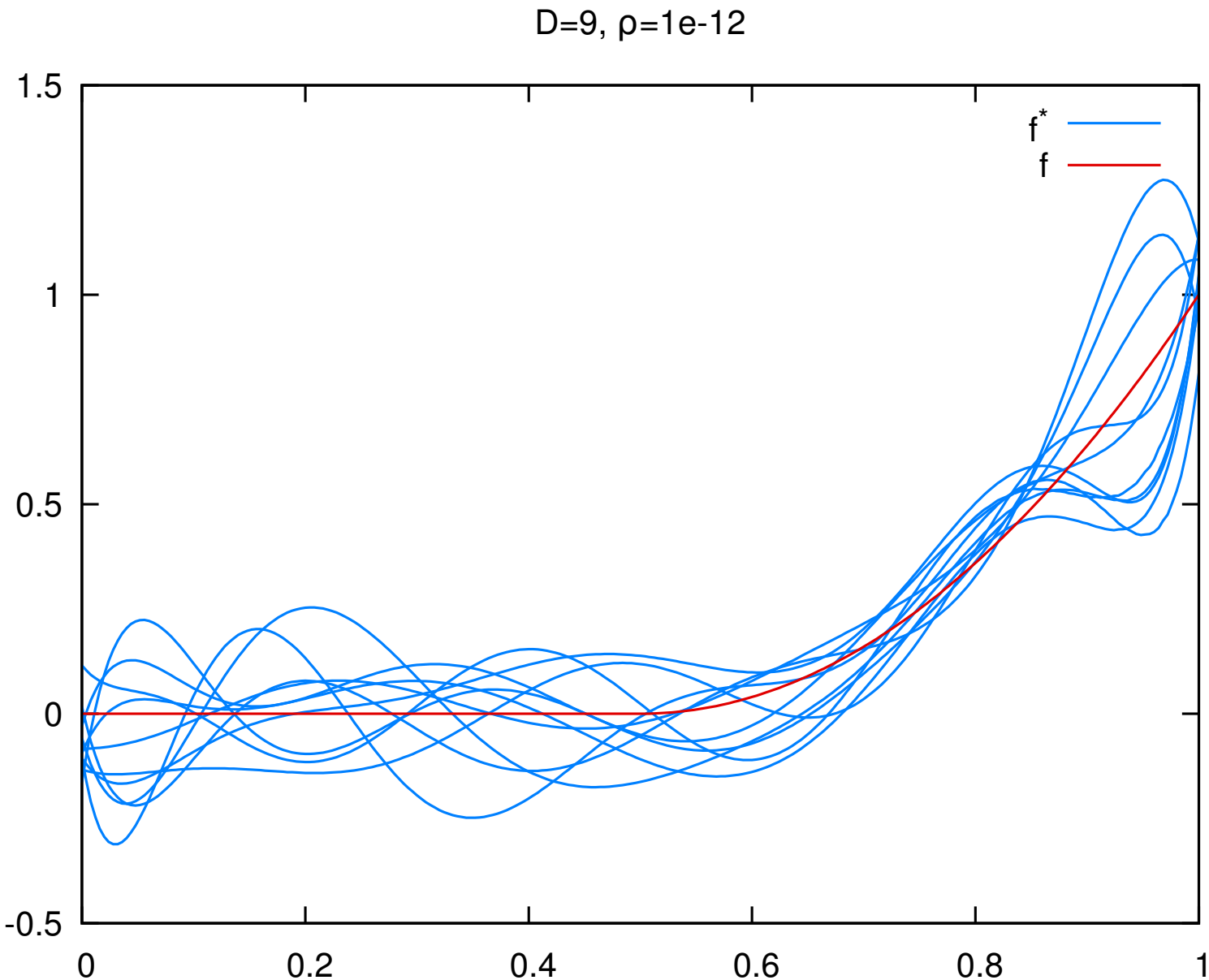

D=9, ρ=1e-7
1.5
1
0.5
0
-0.5
0
0.2
0.4
0.6
0.8
1
f*
f

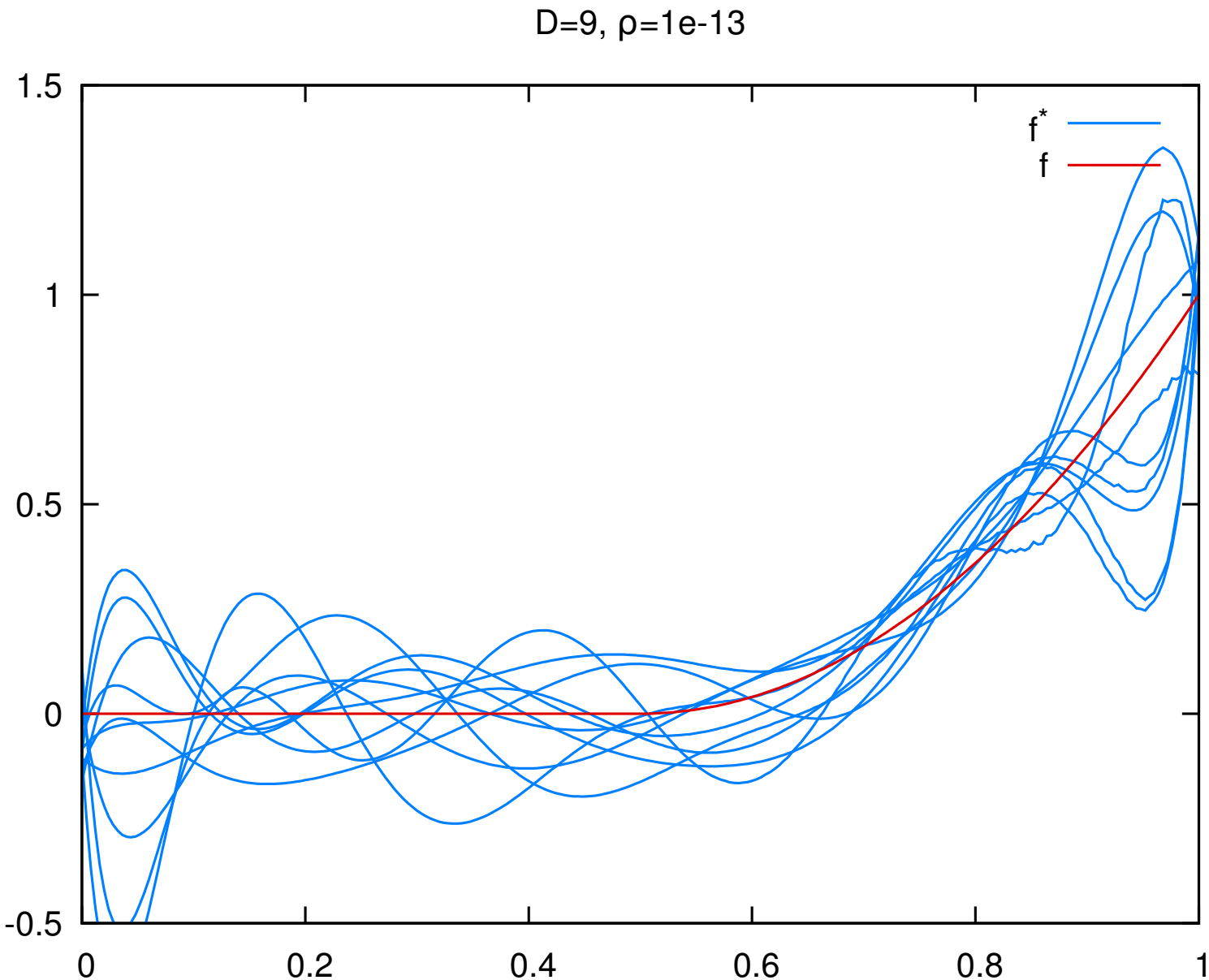

D=9, ρ=1e-8
1.5
1
0.5
0
-0.5
0
0.2
0.4
0.6
0.8
1
f*
f

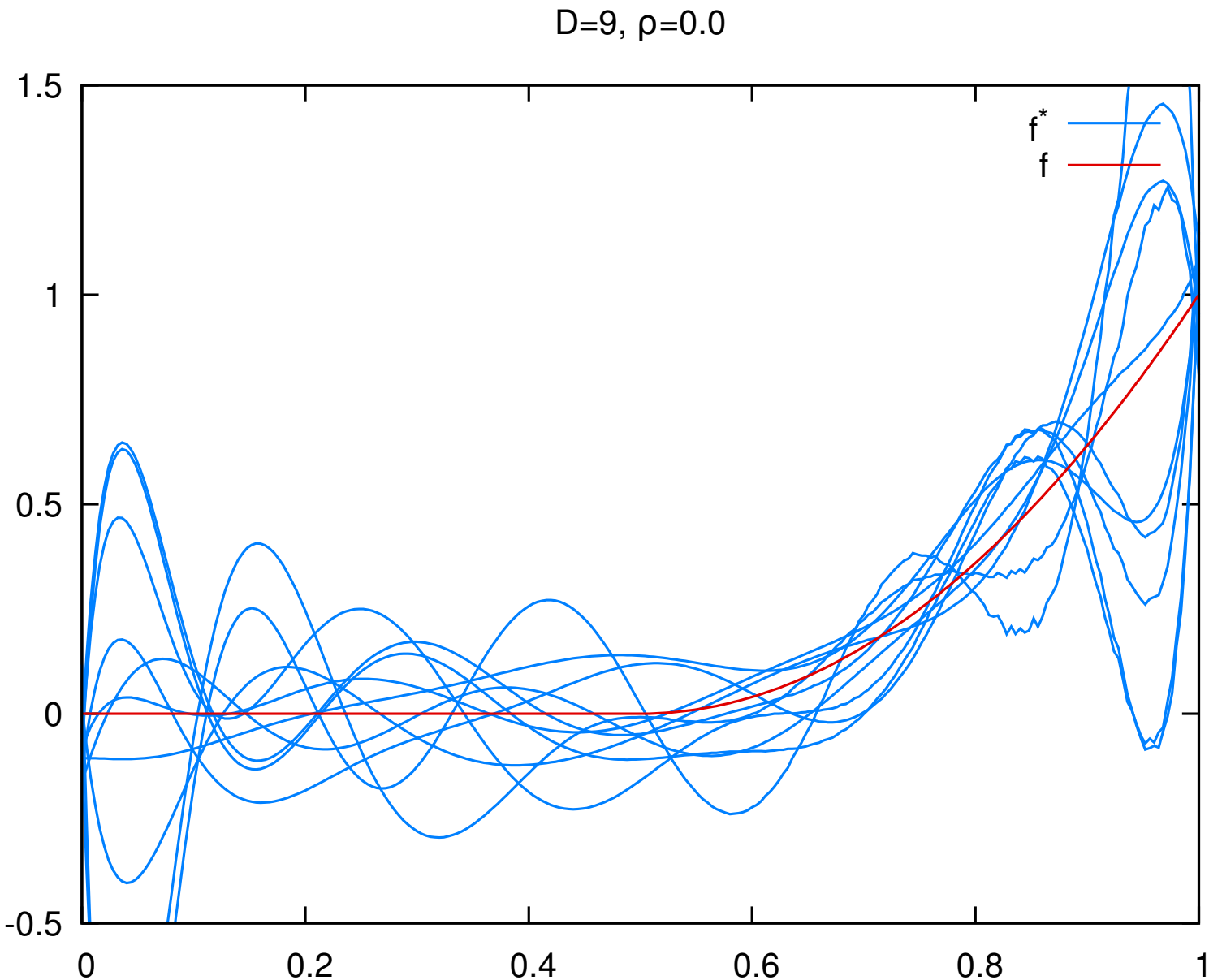

D=9, ρ=1e-9
1.5
1
0.5
0
-0.5
0
0.2
0.4
0.6
0.8
1
f*
f

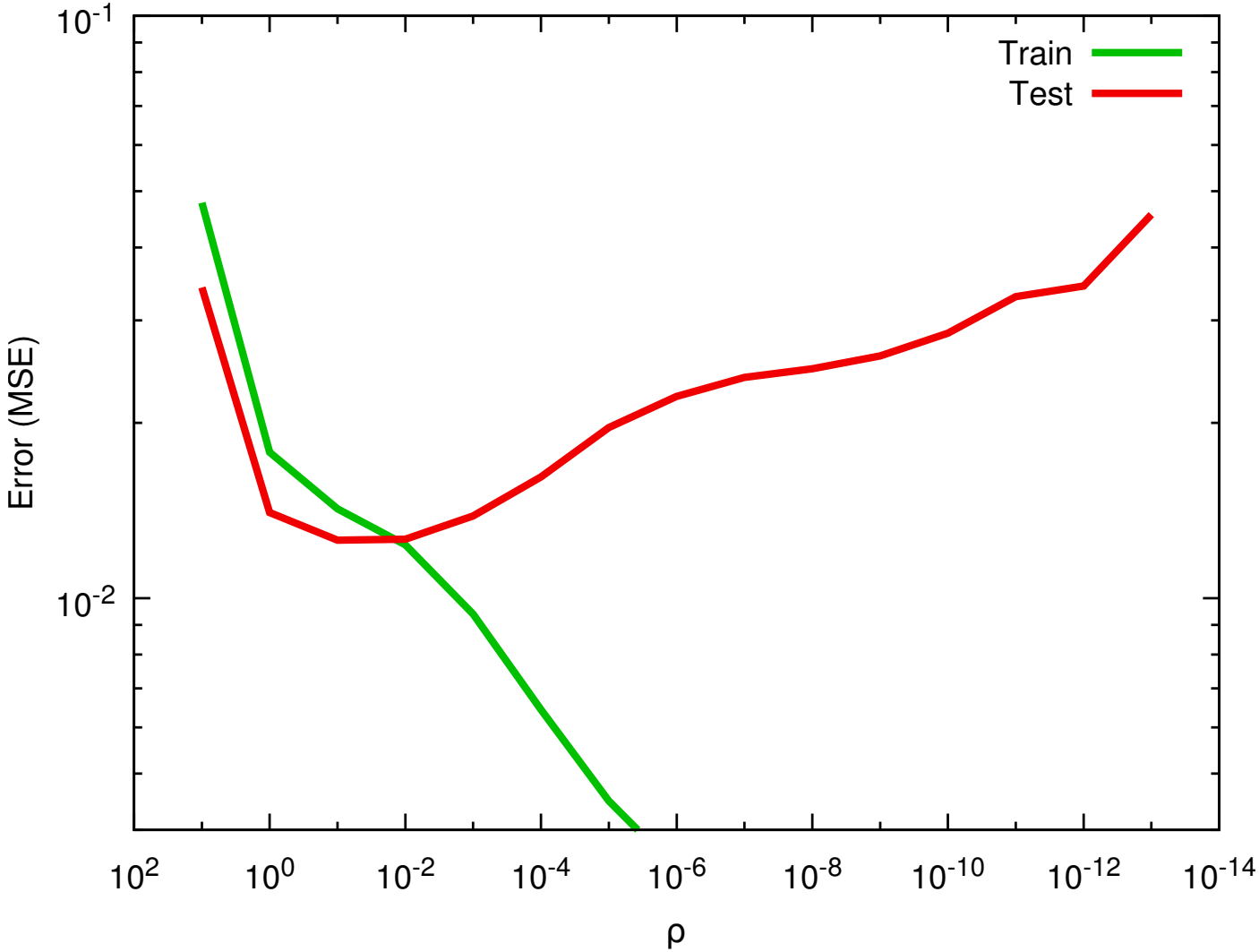

D=9, ρ=1e-10
1.5
1
0.5
0
-0.5
0
0.2
0.4
0.6
0.8
1
f*
f


D=9, ρ=1e-11
1.5
1
0.5
0
-0.5
0
0.2
0.4
0.6
0.8
1
f*
f


D=9, ρ=1e-12
1.5
1
0.5
0
-0.5
0
0.2
0.4
0.6
0.8
1
f*
f


D=9, ρ=1e-13
1.5
1
0.5
0
-0.5
0
0.2
0.4
0.6
0.8
1
f*
f


D=9, ρ=0.0
1.5
1
0.5
0
-0.5
0
0.2
0.4
0.6
0.8
1
f*
f


Train
Test
Error (MSE)
10^{-1}
10^{-2}
10^{2}
10^{0}
10^{-2}
10^{-4}
10^{-6}
10^{-8}
10^{-10}
10^{-12}
10^{-14}
ρ

We define the **capacity** of a set of predictors as its ability to model an arbitrary functional. This is a vague definition, difficult to make formal.

We define the **capacity** of a set of predictors as its ability to model an arbitrary functional. This is a vague definition, difficult to make formal.

A mathematically precise notion is the Vapnik–Chervonenkis dimension of a set of functions, which, in the Binary classification case, is the cardinality of the largest set that can be labeled arbitrarily (Vapnik, 1995).

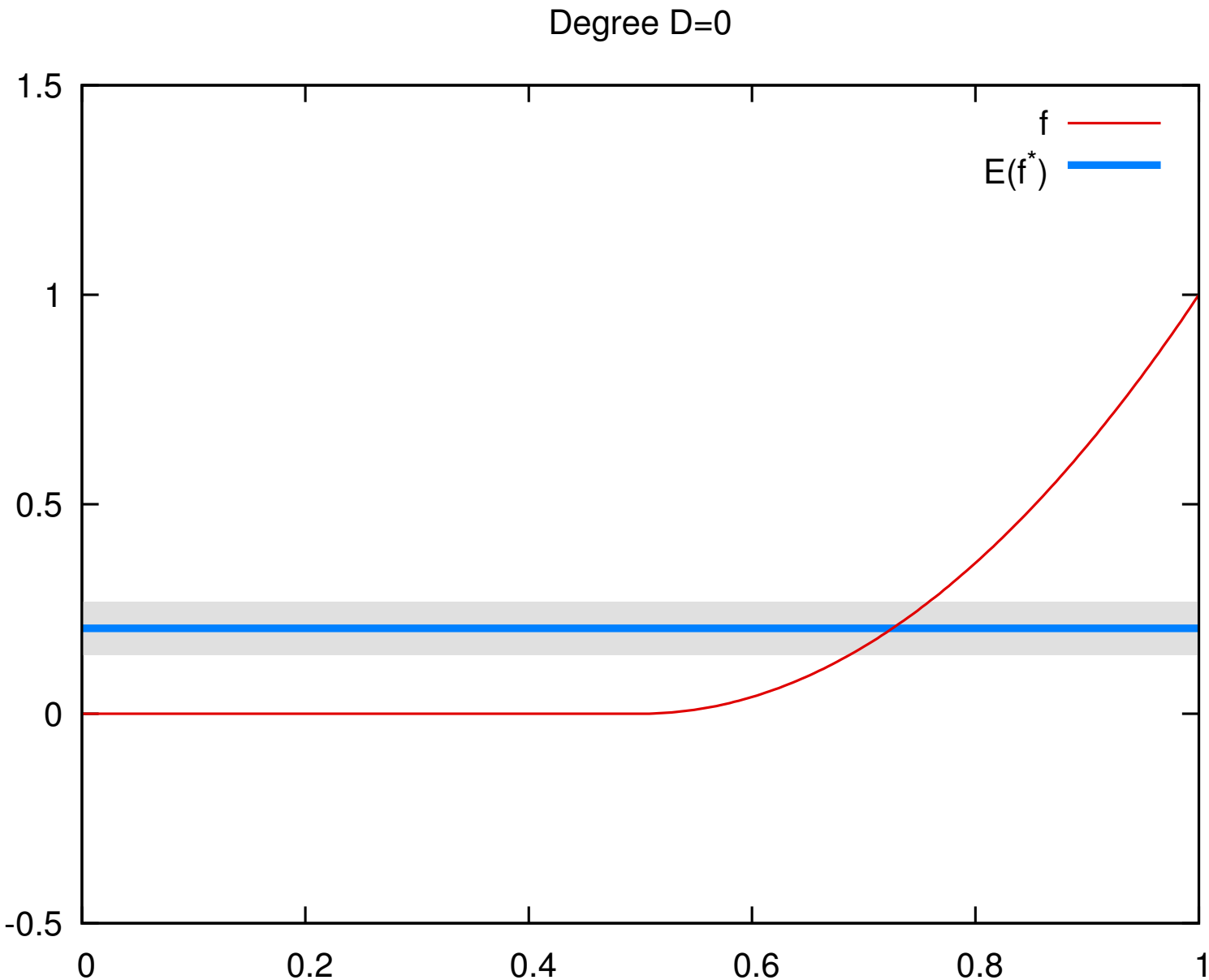
It is a very powerful concept, but is poorly adapted to neural networks. We will not say more about it in this course.

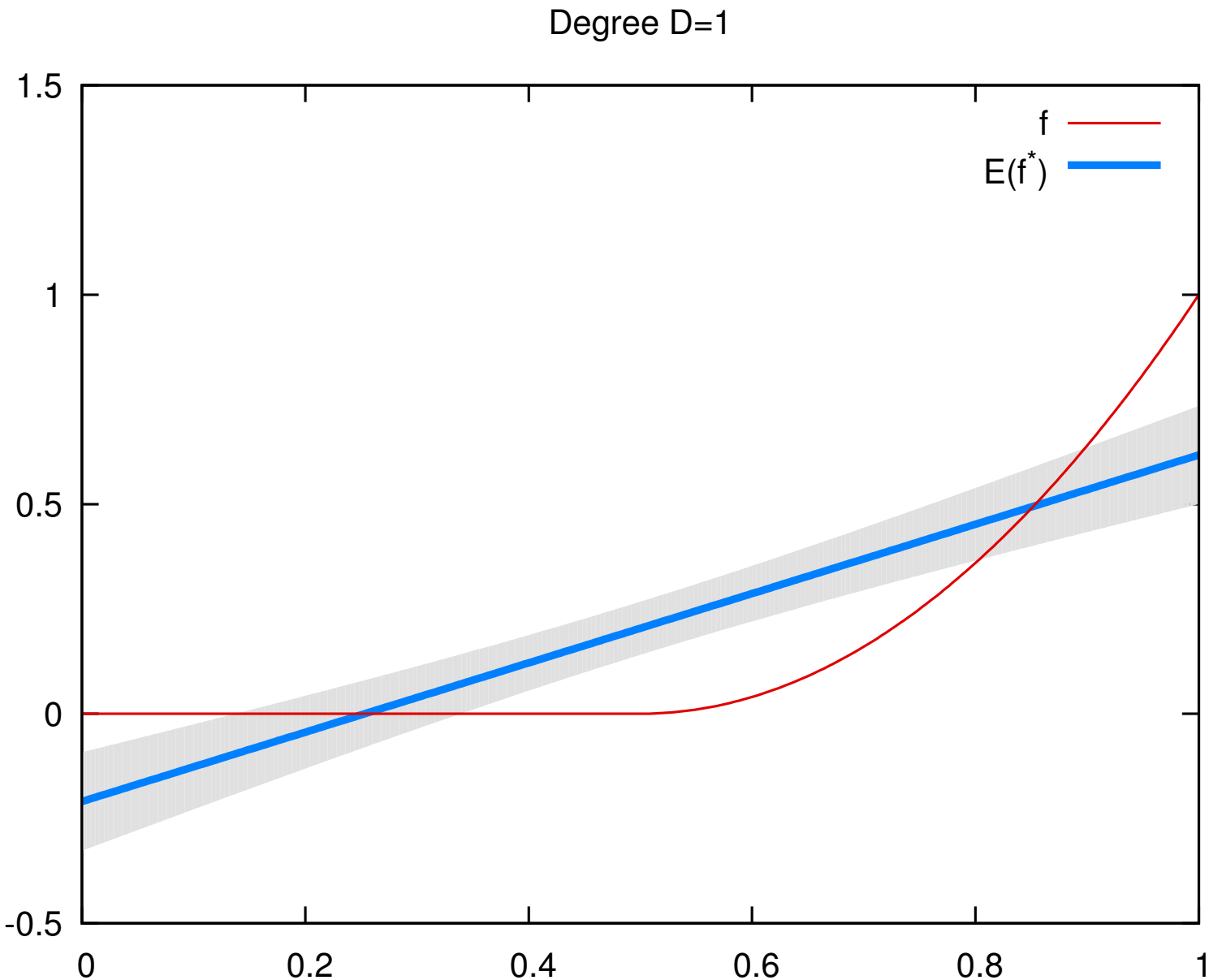
Although the capacity is hard to define precisely, it is quite clear in practice how to modulate it for a given class of models.

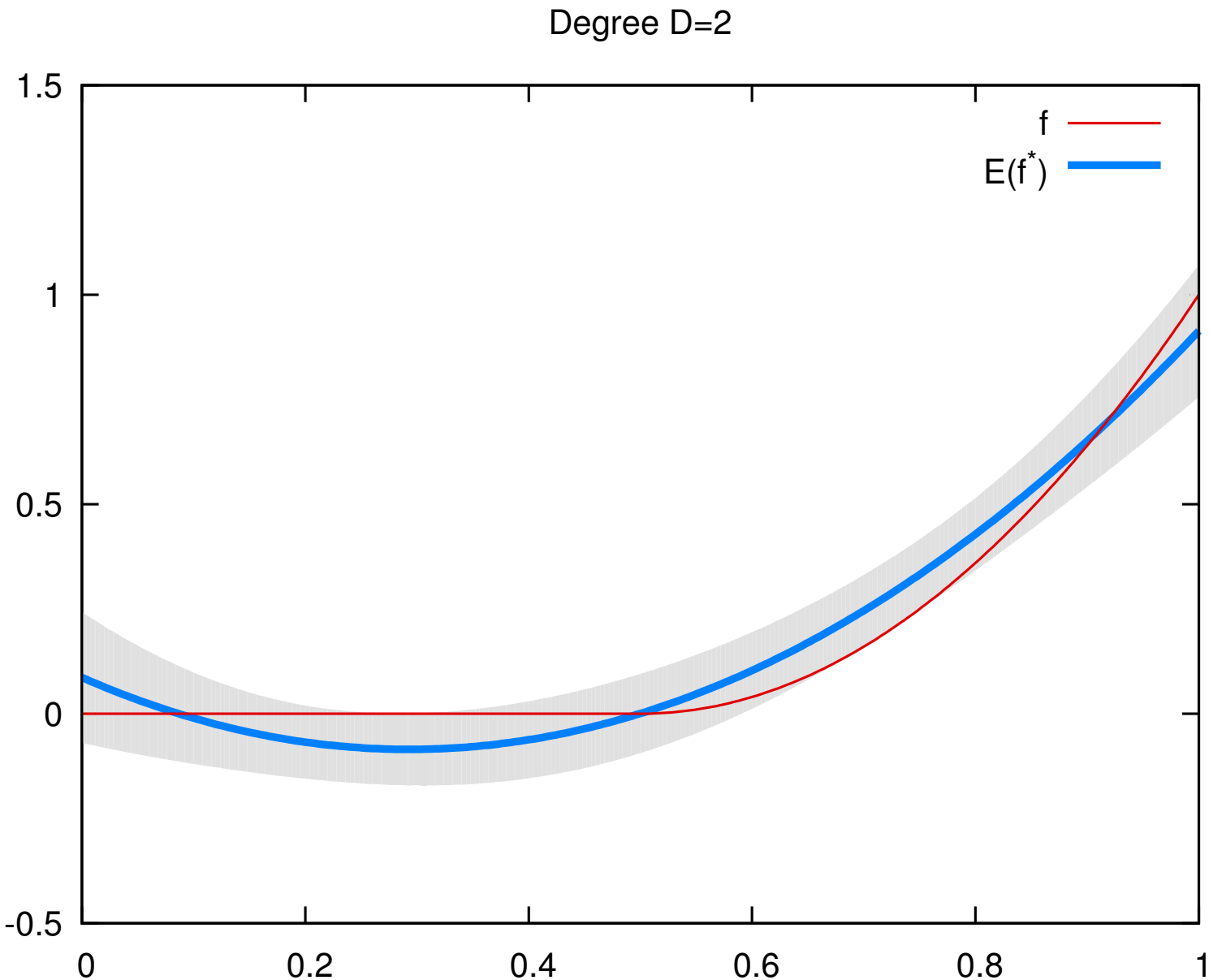
In particular one can control over-fitting either by

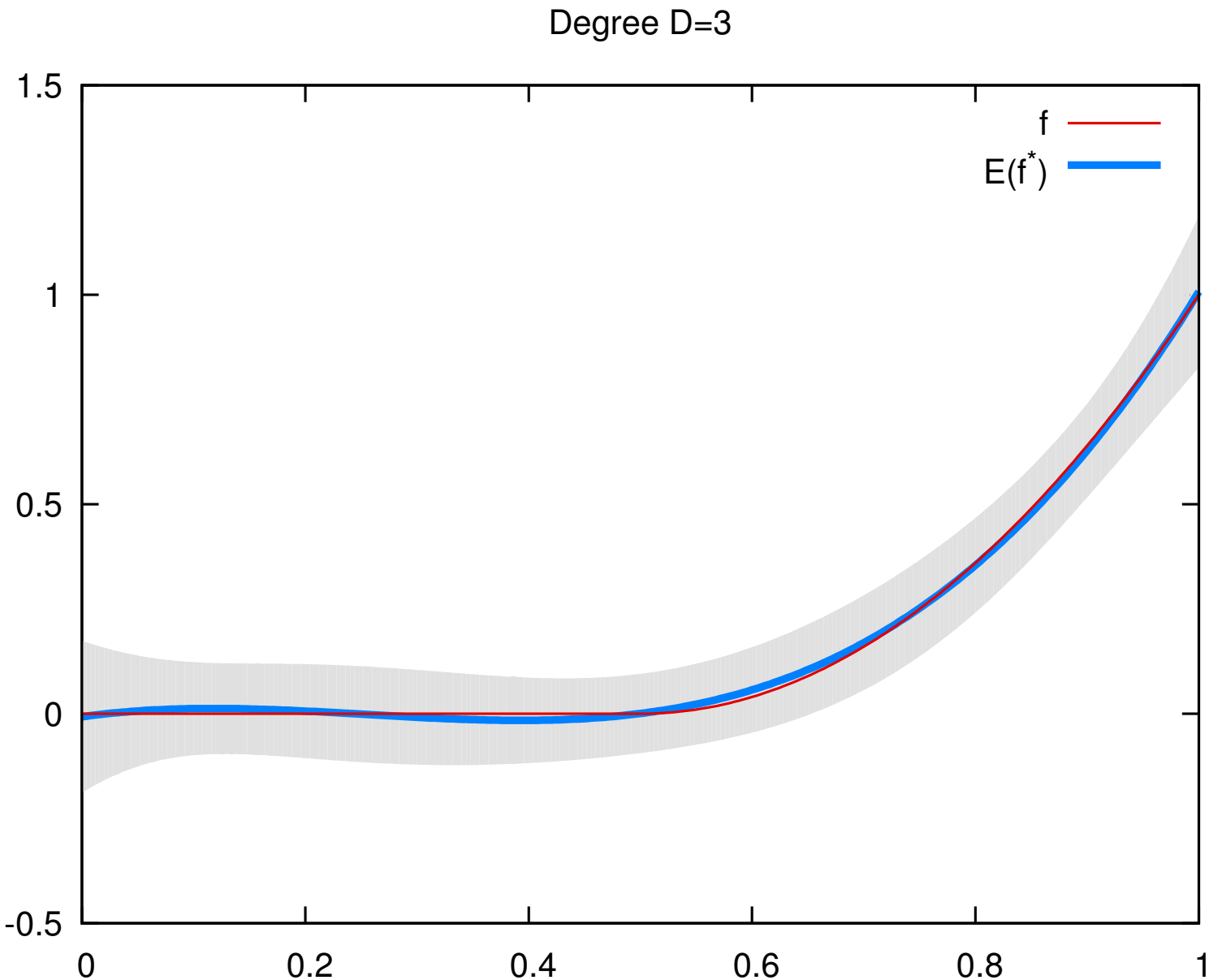
- Impoverishing the space $\mathscr{F}$ (less functionals, early stopping)
- Make the choice of f^* less dependent on data (penalty on coefficients, margin maximization, ensemble methods)

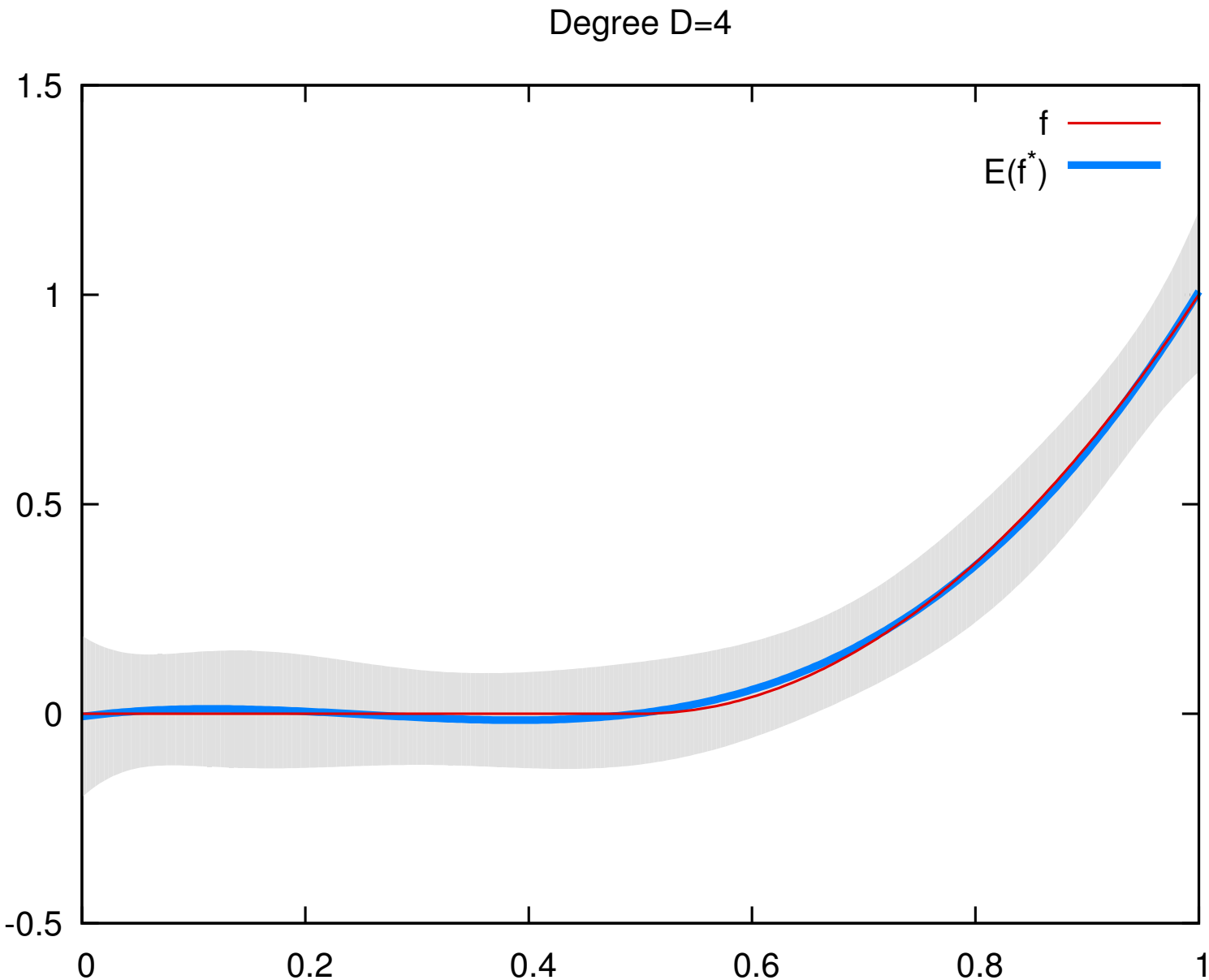
Bias-variance dilemma

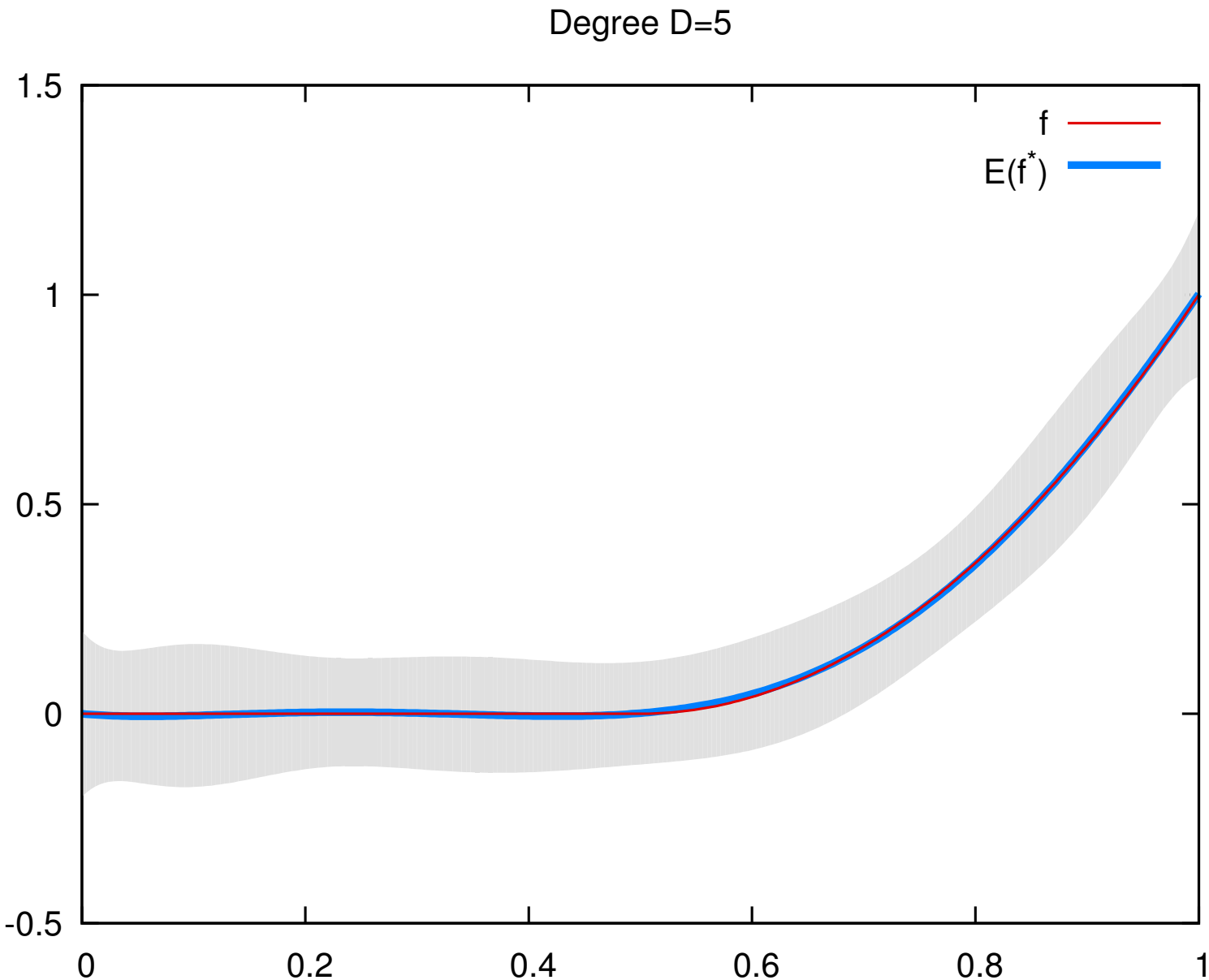

Degree D=0
1.5
1
0.5
0
-0.5
0
0.2
0.4
0.6
0.8
1
f
E(f*)

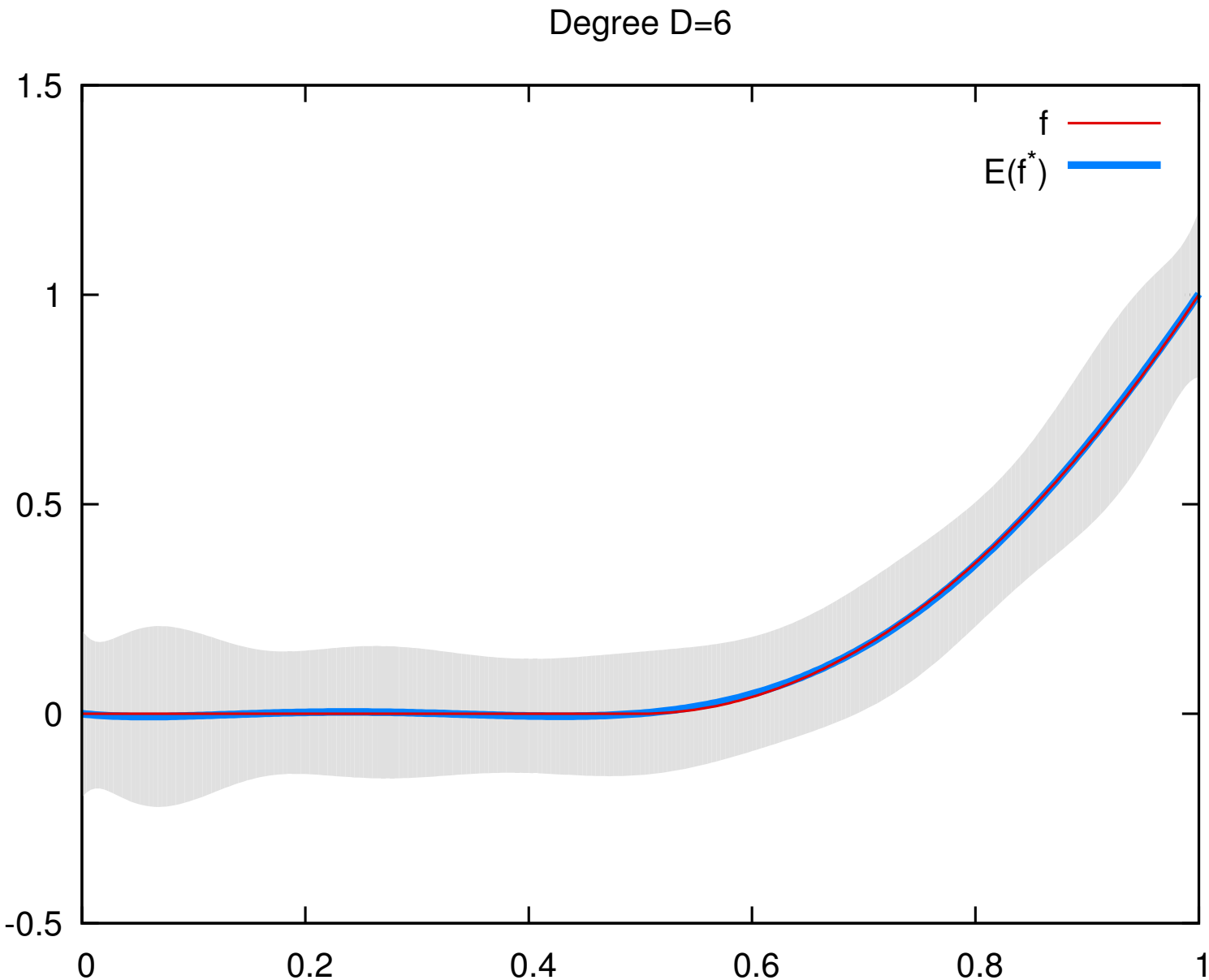

Degree D=1
1.5
1
0.5
0
-0.5
0
0.2
0.4
0.6
0.8
1
f
E(f*)

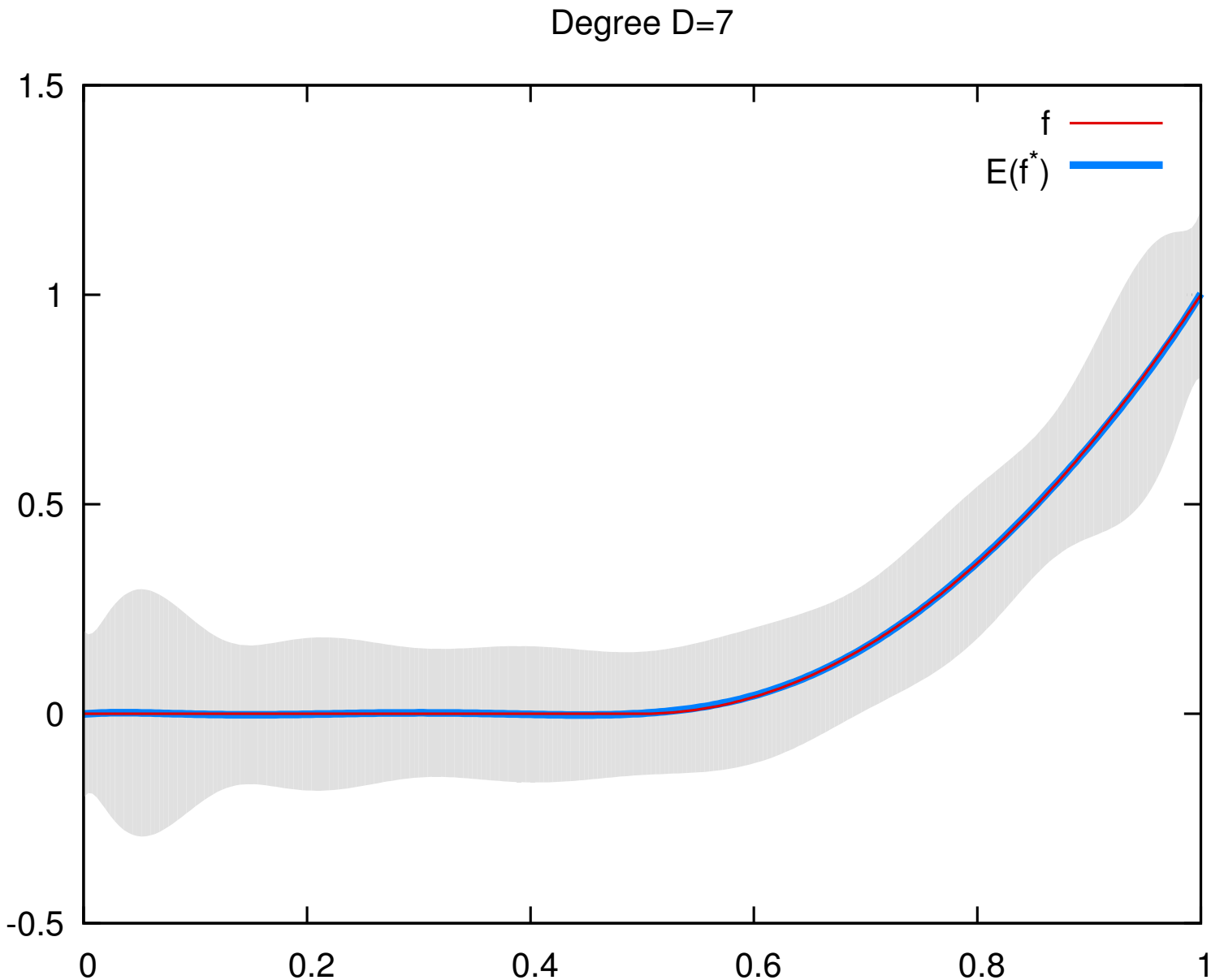

Degree D=2
1.5
1
0.5
0
-0.5
0
0.2
0.4
0.6
0.8
1
f
E(f*)

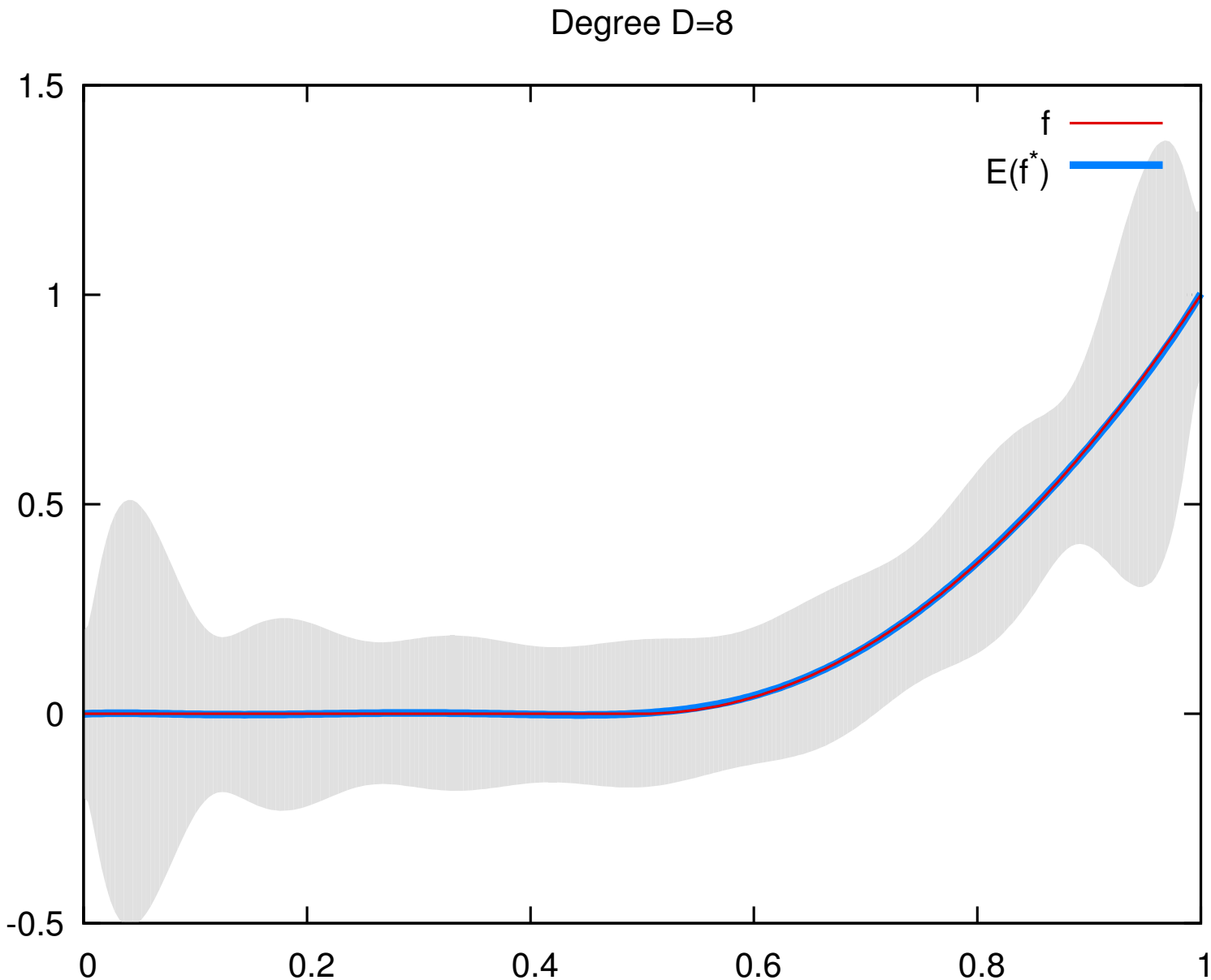

Degree D=3
1.5
1
0.5
0
-0.5
0
0.2
0.4
0.6
0.8
1
f
E(f*)

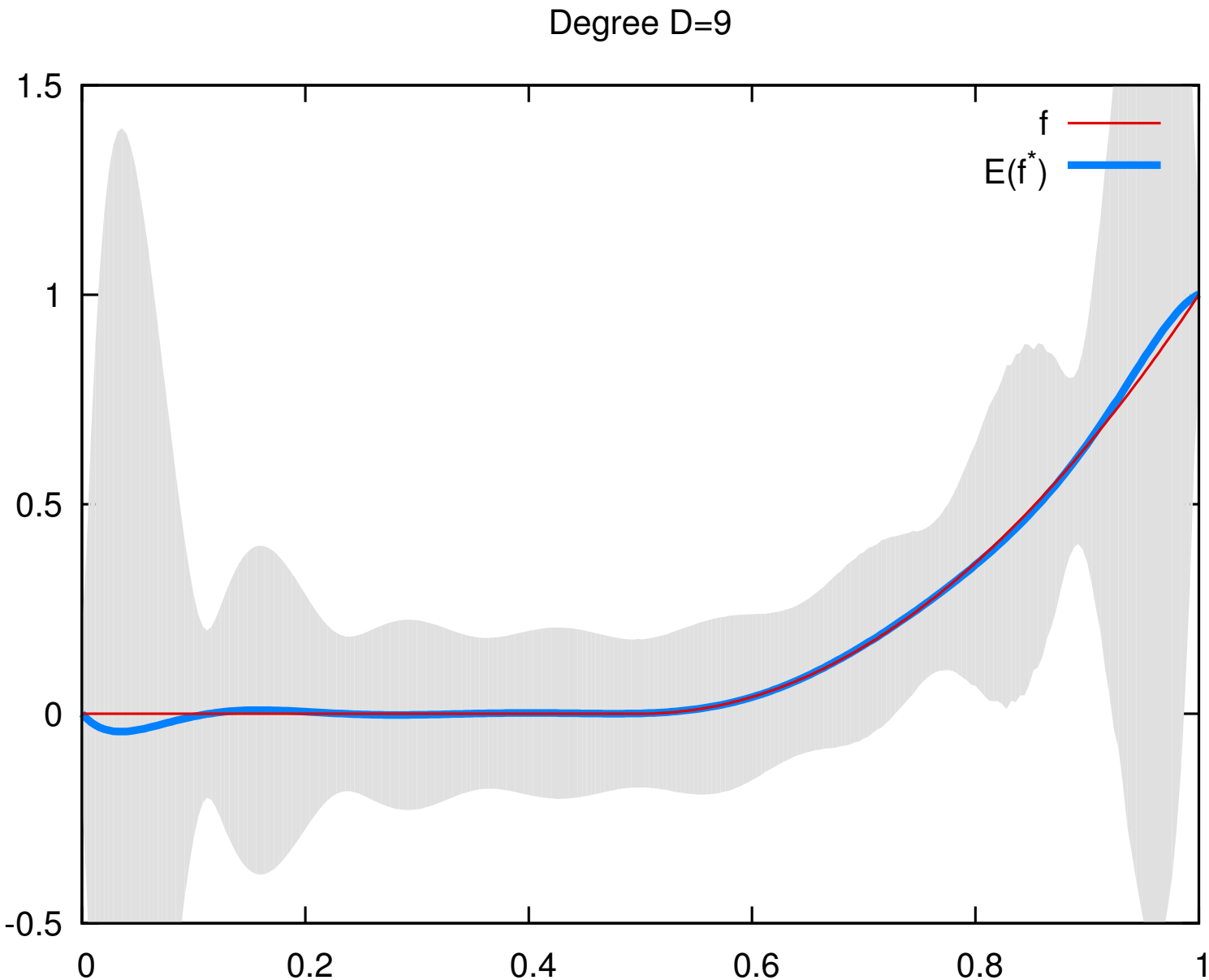

Degree D=4
1.5
1
0.5
0
-0.5
0
0.2
0.4
0.6
0.8
1
f
E(f*)

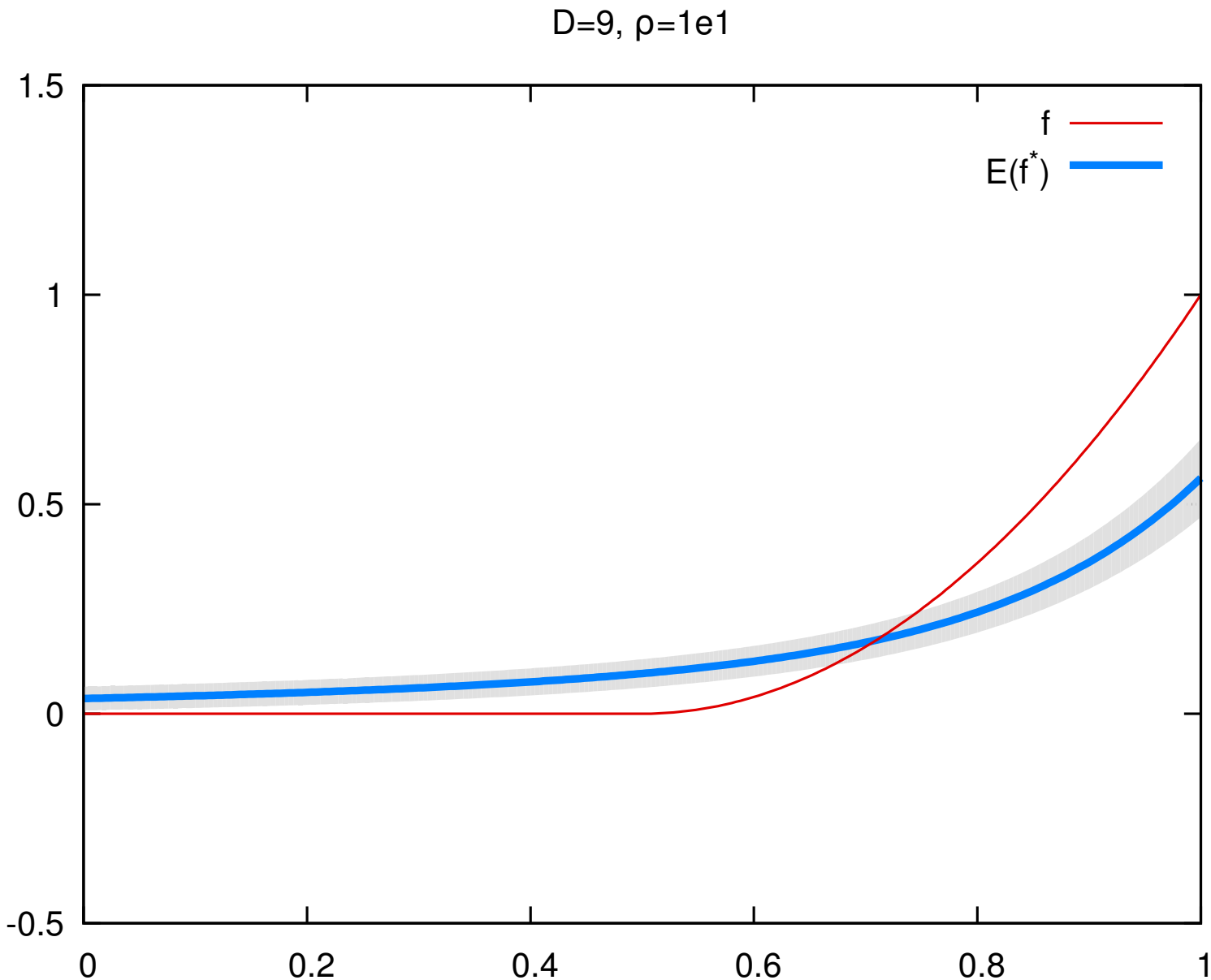

Degree D=5
1.5
1
0.5
0
-0.5
0
0.2
0.4
0.6
0.8
1
f
E(f*)

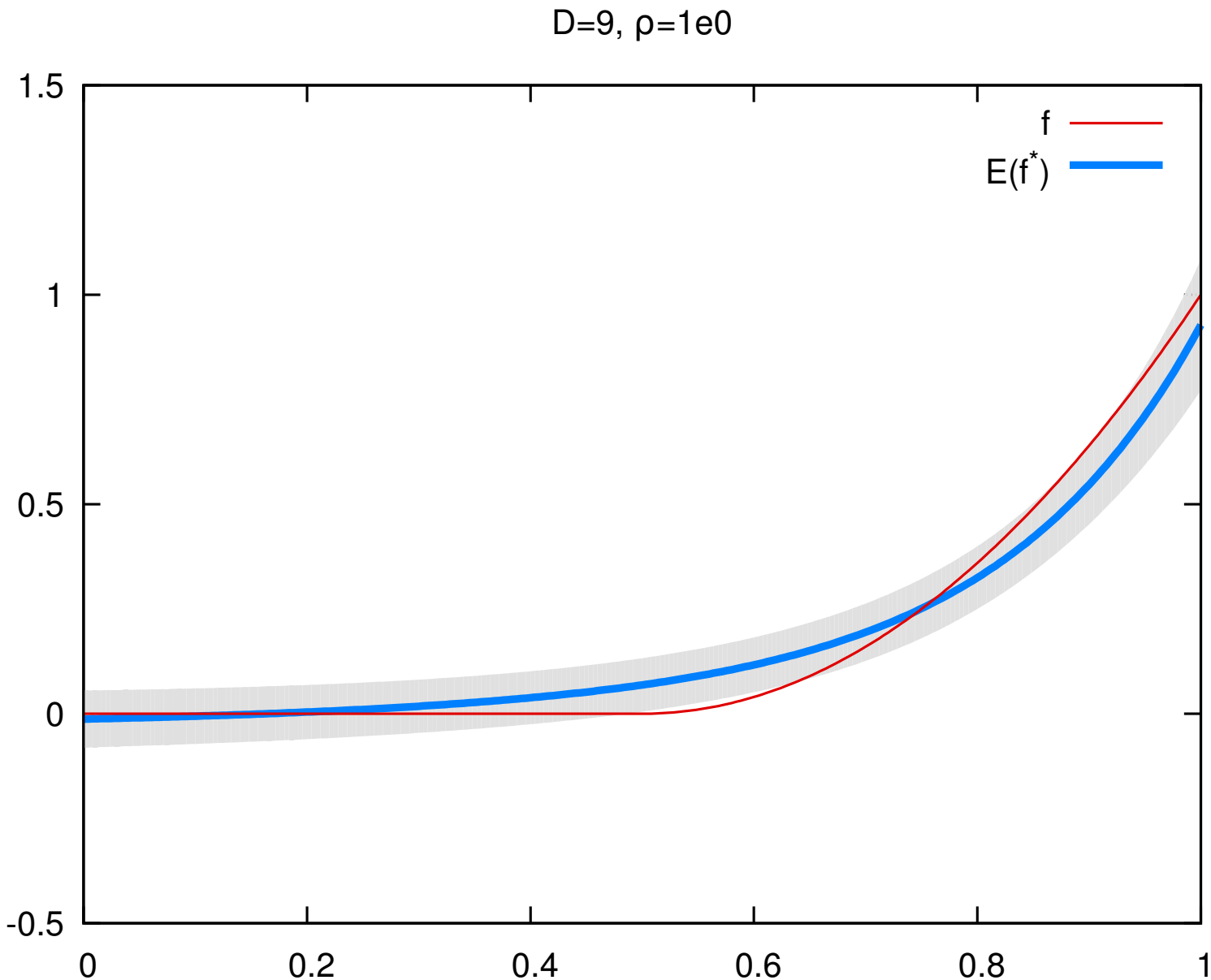

Degree D=6
1.5
1
0.5
0
-0.5
0
0.2
0.4
0.6
0.8
1
f
E(f*)

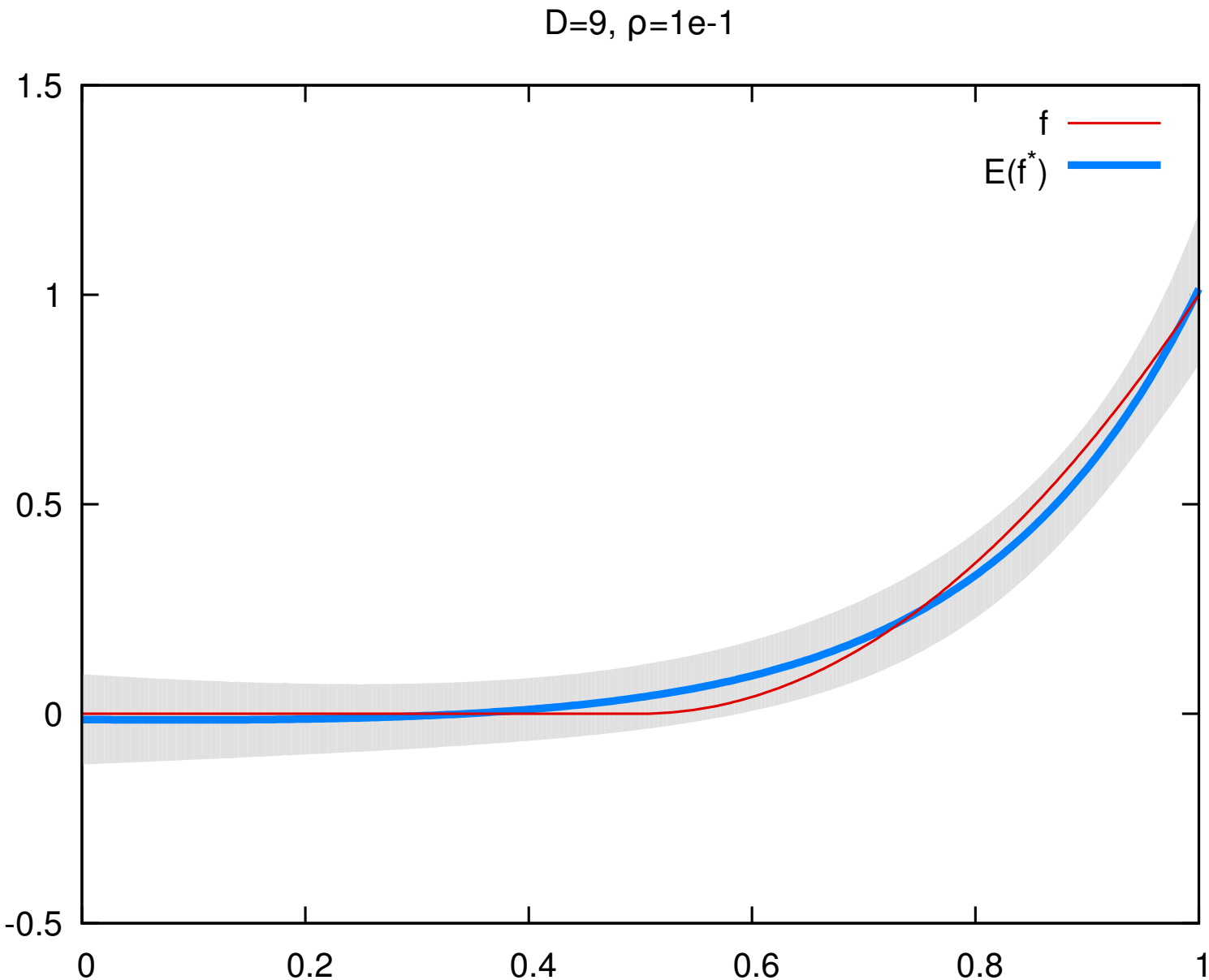

Degree D=7
1.5
1
0.5
0
-0.5
0
0.2
0.4
0.6
0.8
1
f
E(f*)

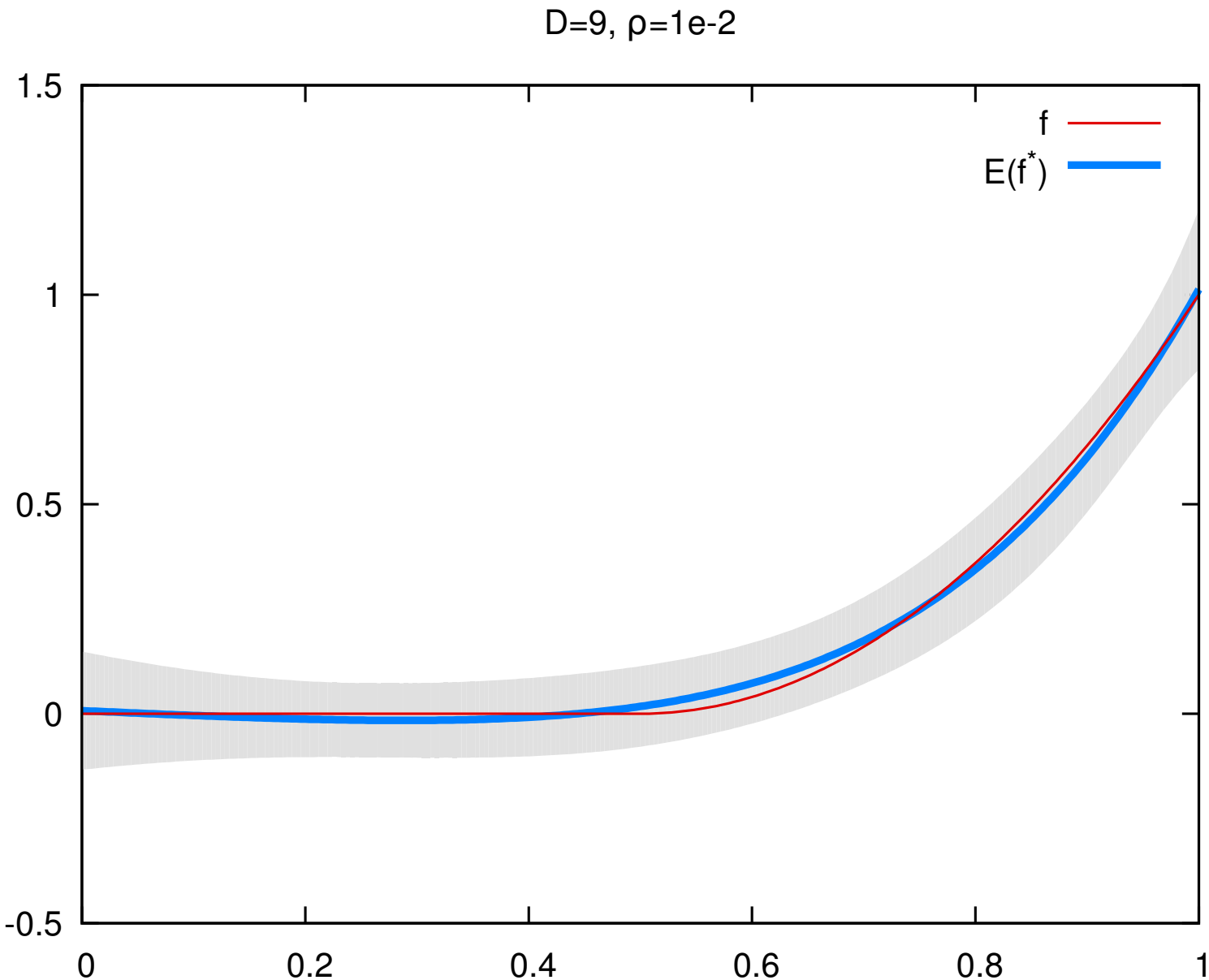

Degree D=8
1.5
1
0.5
0
-0.5
0
0.2
0.4
0.6
0.8
1
f
E(f*)

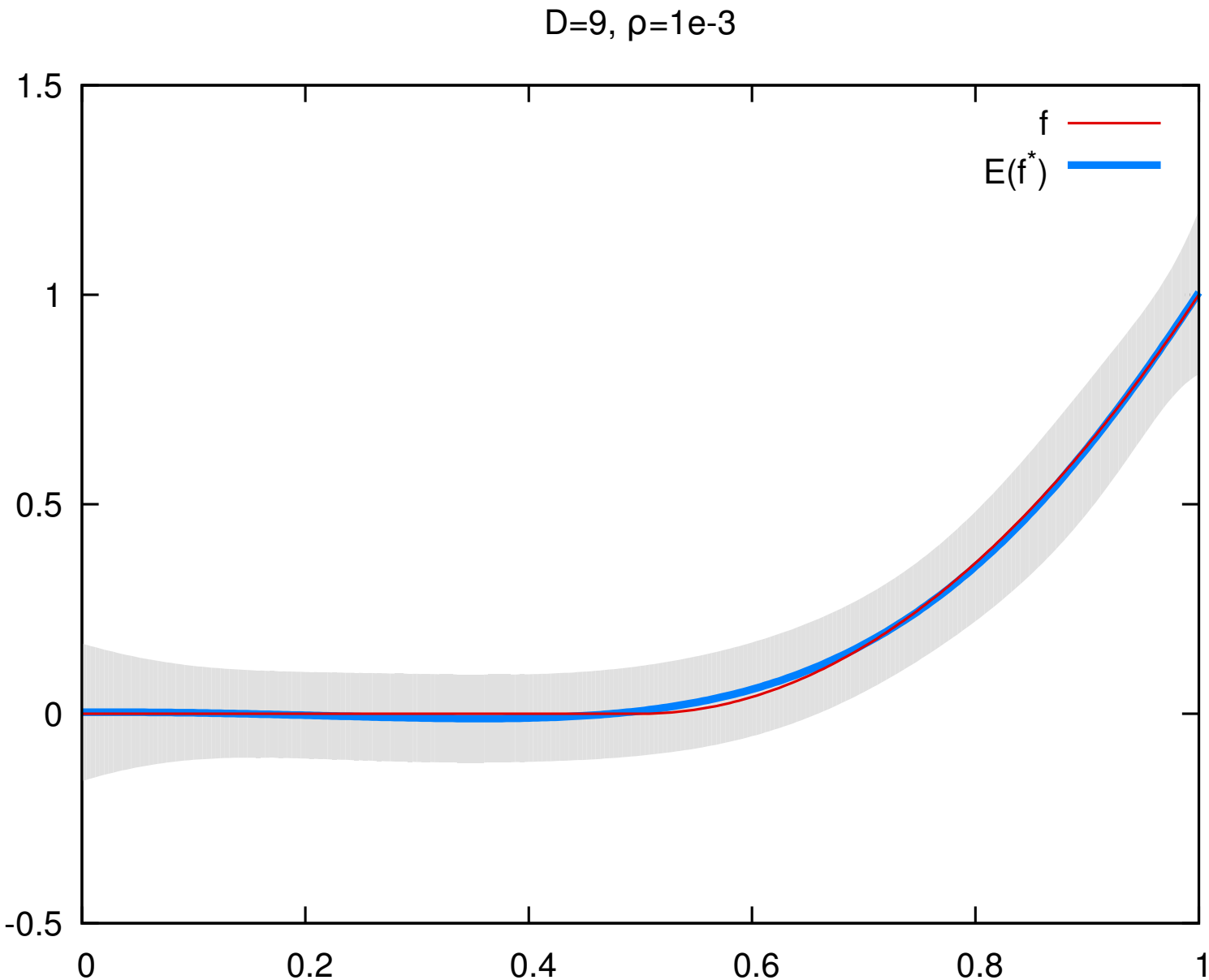

Degree D=9
1.5
1
0.5
0
-0.5
0
0.2
0.4
0.6
0.8
1
f
E(f*)

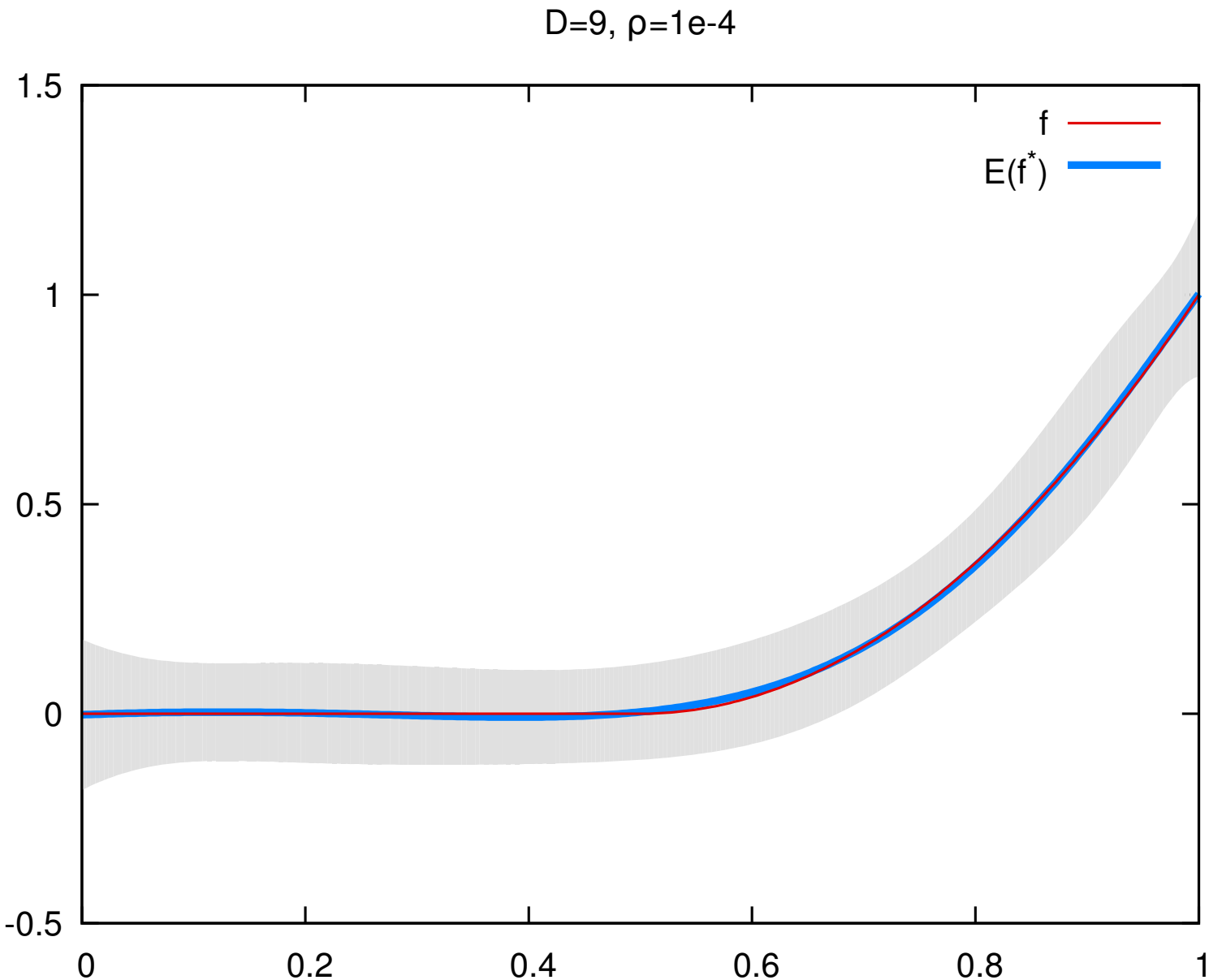

D=9, ρ=1e1
1.5
1
0.5
0
-0.5
0
0.2
0.4
0.6
0.8
1
f
E(f*)

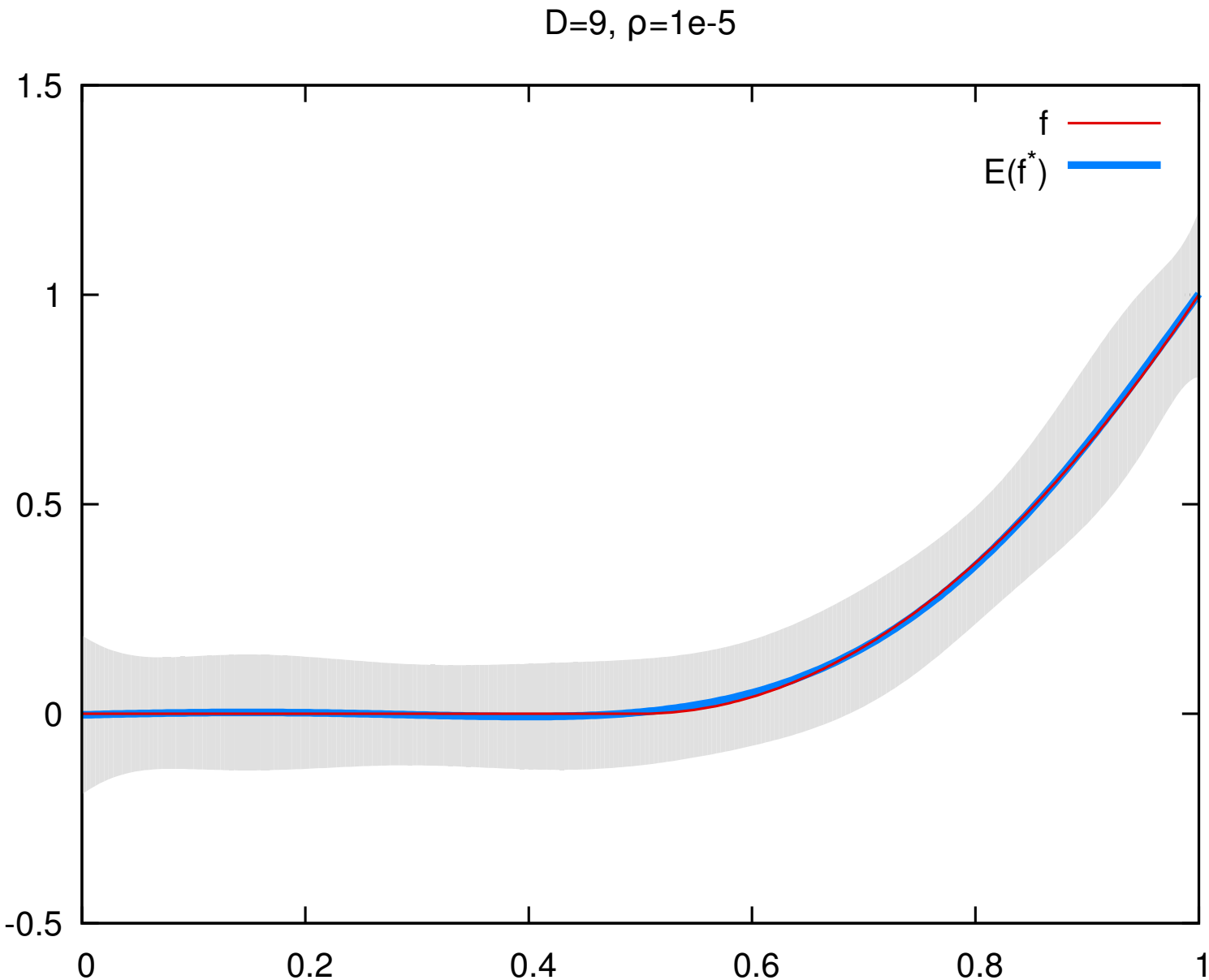

D=9, ρ=1e0
1.5
1
0.5
0
-0.5
0
0.2
0.4
0.6
0.8
1
f
E(f*)

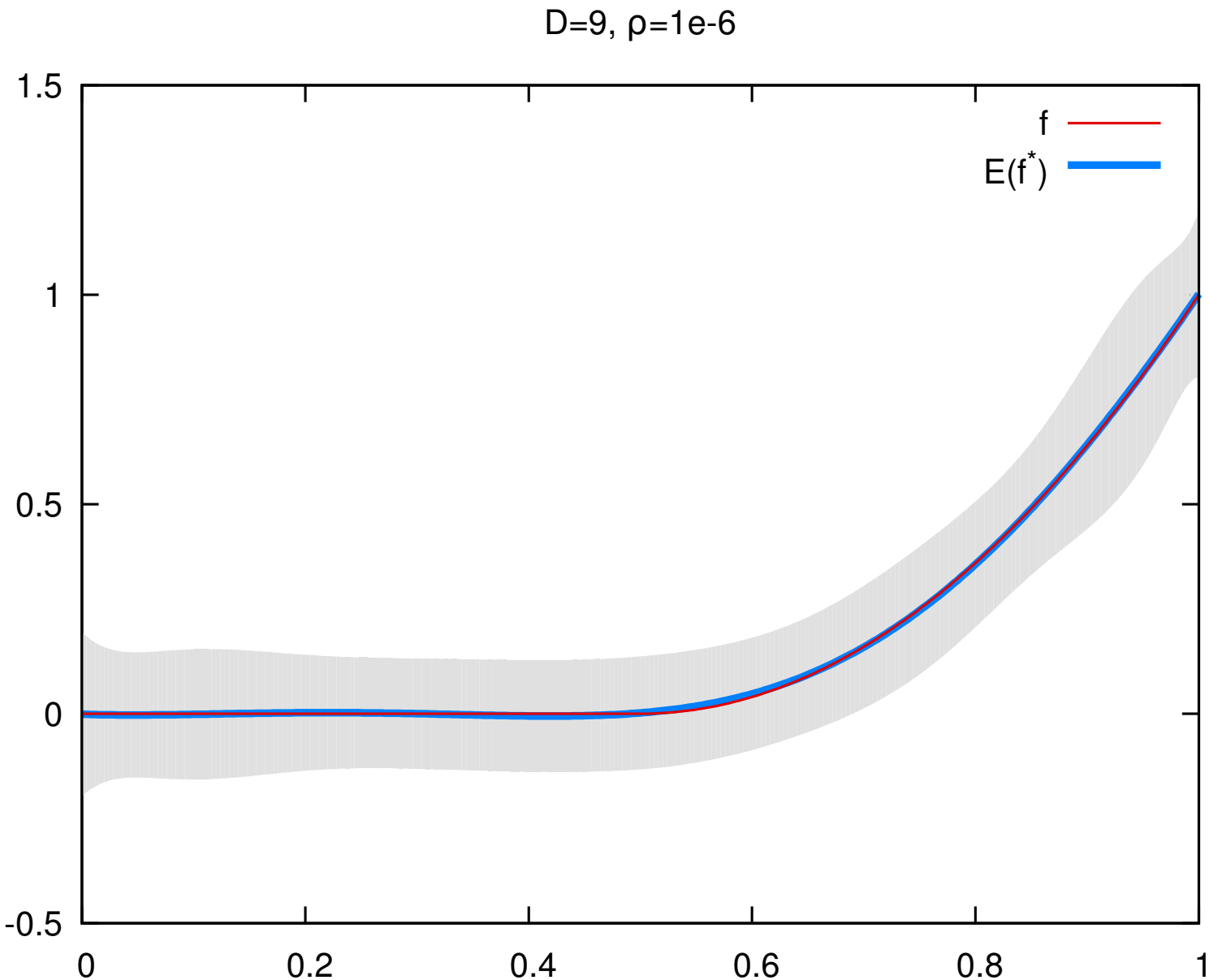

D=9, ρ=1e-1
1.5
1
0.5
0
-0.5
0
0.2
0.4
0.6
0.8
1
f
E(f*)

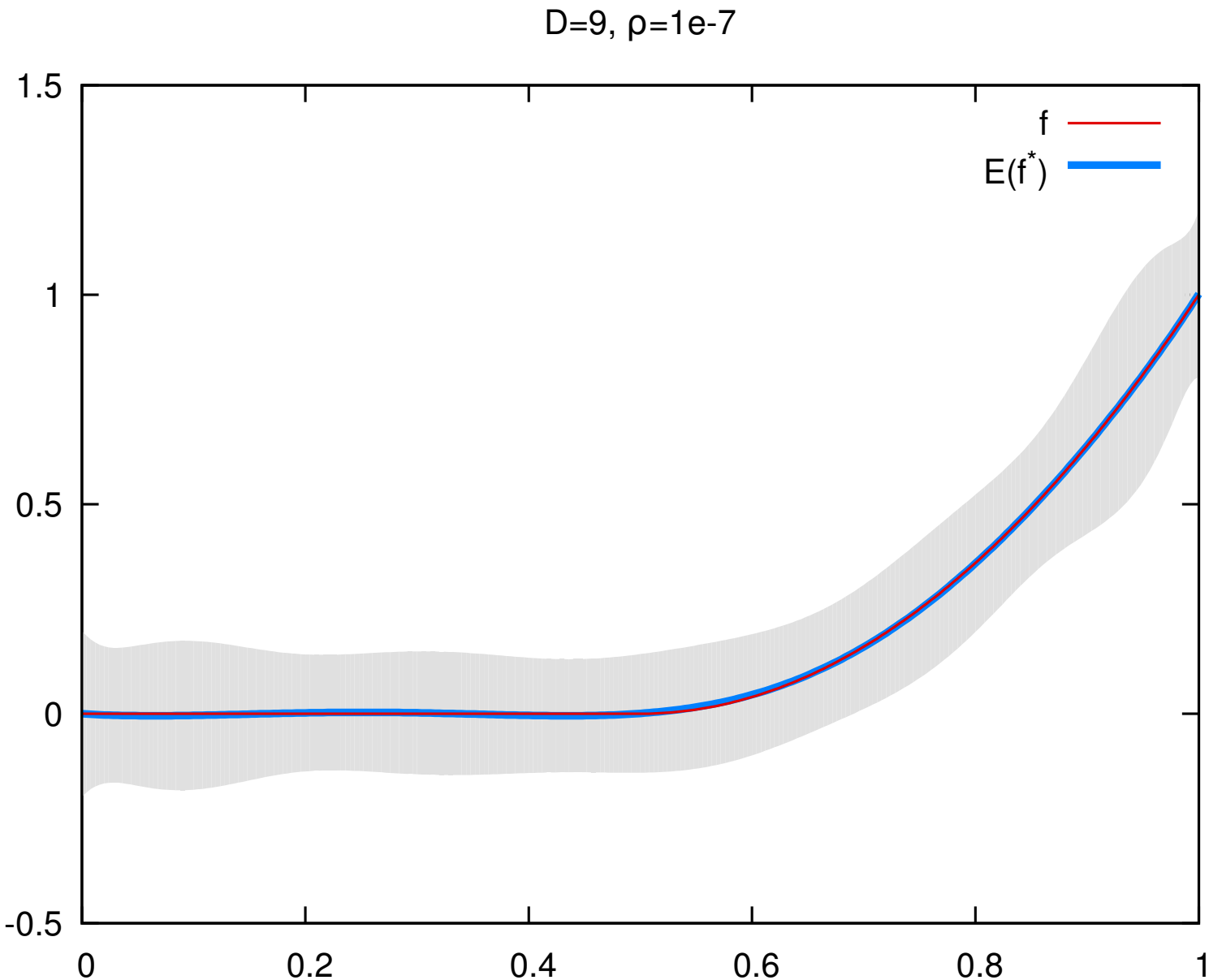

D=9, ρ=1e-2
1.5
1
0.5
0
-0.5
0
0.2
0.4
0.6
0.8
1
f
E(f*)

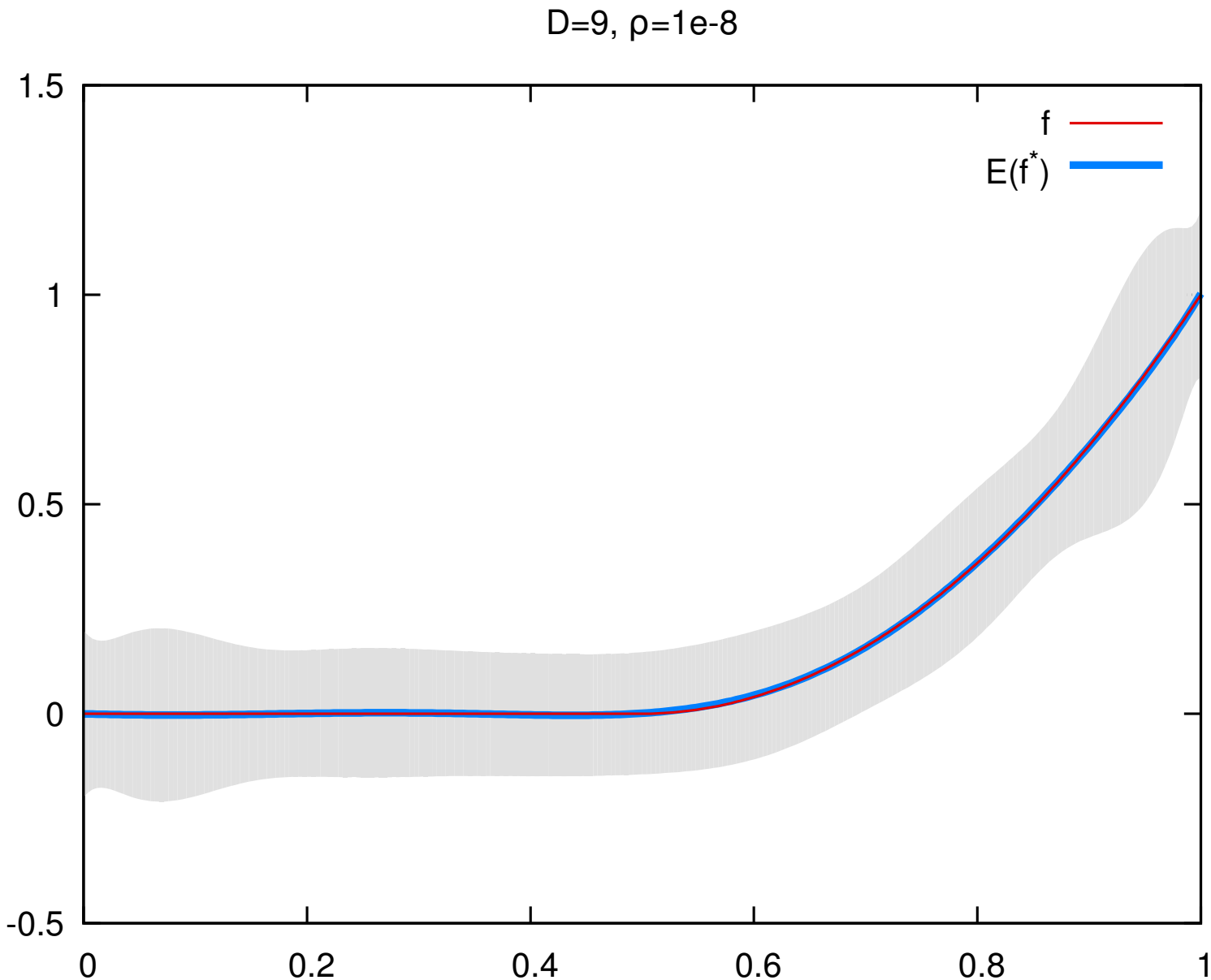

D=9, ρ=1e-3
1.5
1
0.5
0
-0.5
0
0.2
0.4
0.6
0.8
1
f
E(f*)

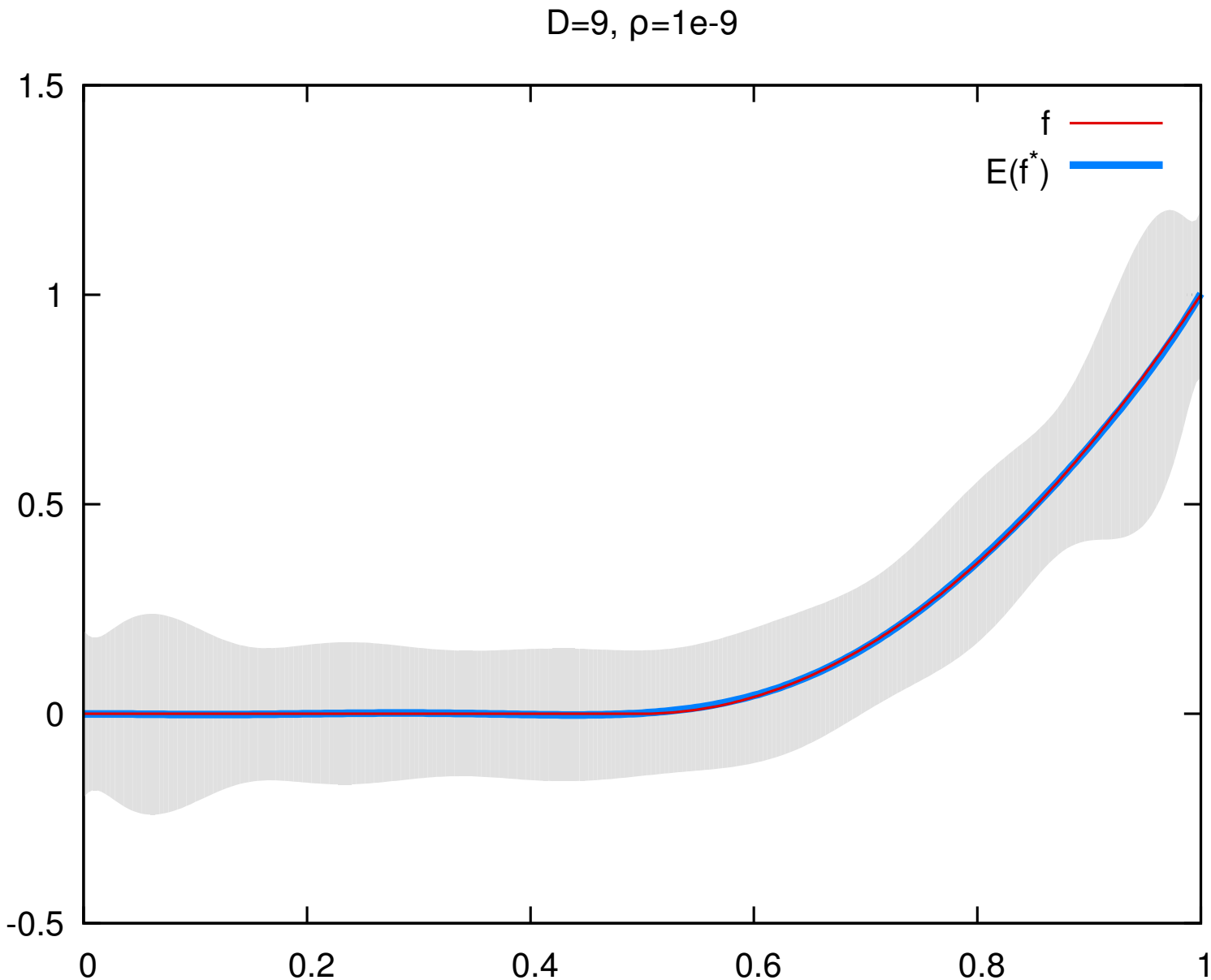

D=9, ρ=1e-4
f
E(f*)
1.5
1
0.5
0
-0.5
0
0.2
0.4
0.6
0.8
1

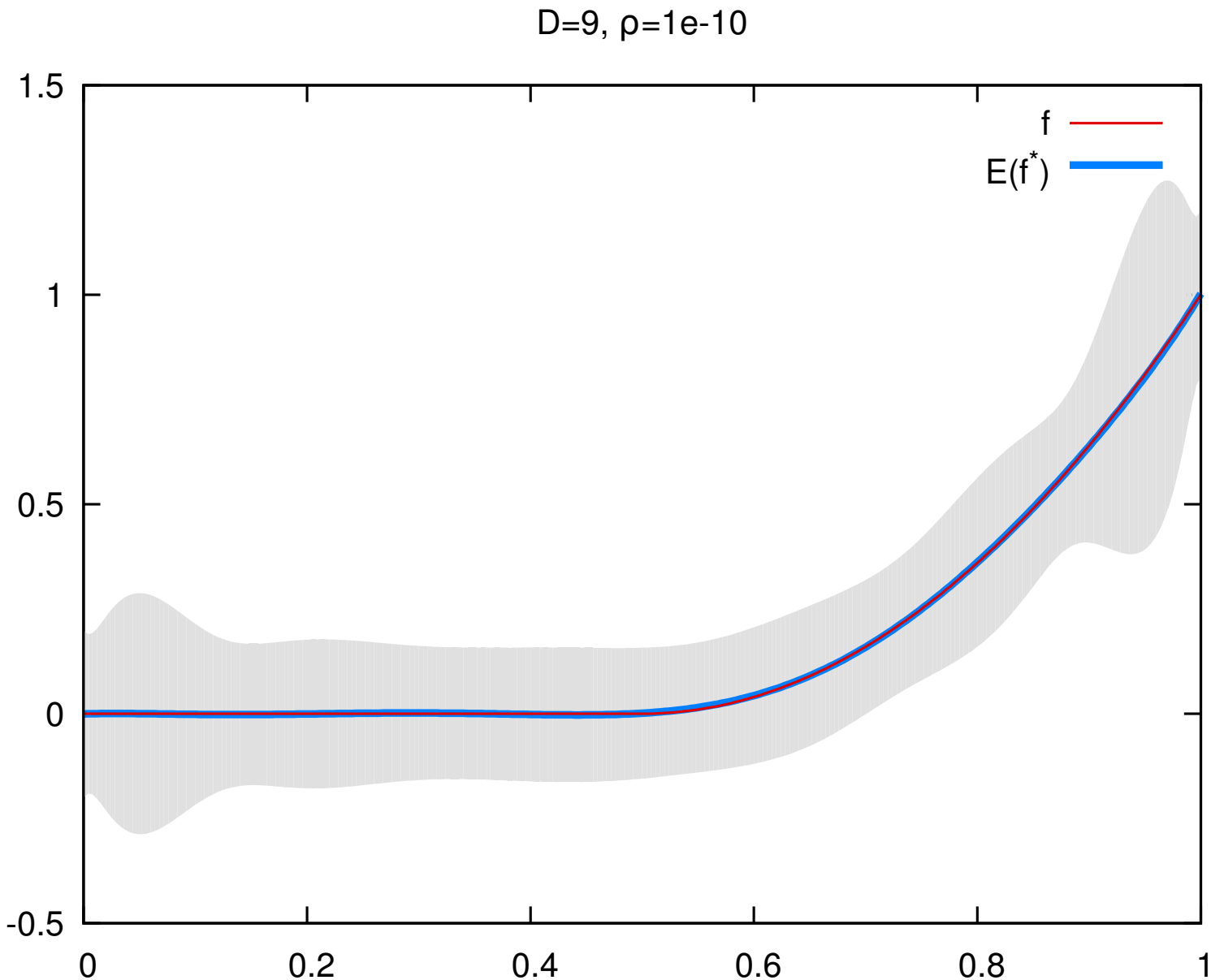

D=9, ρ=1e-5
f
E(f*)
1.5
1
0.5
0
-0.5
0
0.2
0.4
0.6
0.8
1

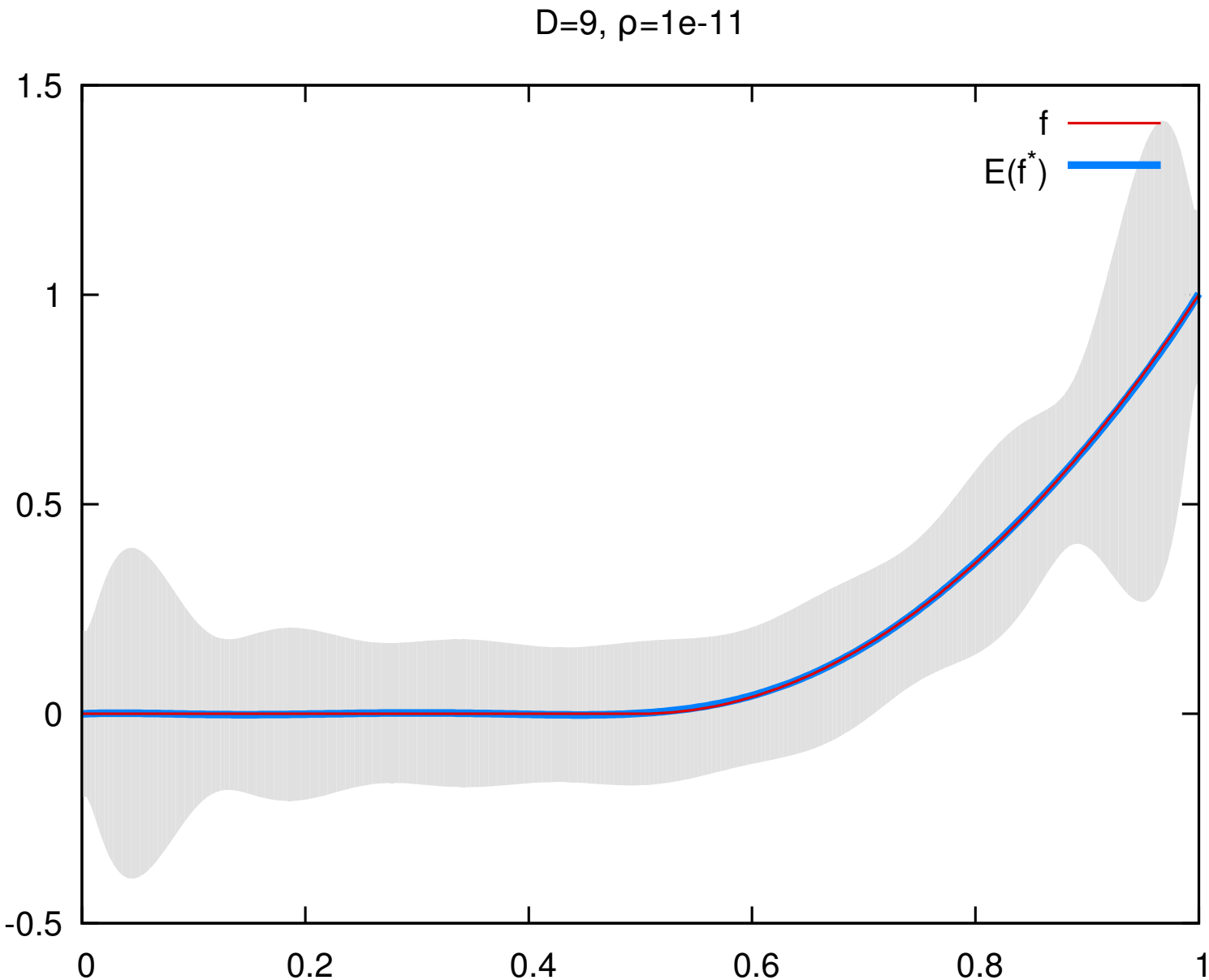

D=9, ρ=1e-6
1.5
1
0.5
0
-0.5
0
0.2
0.4
0.6
0.8
1
f
E(f*)

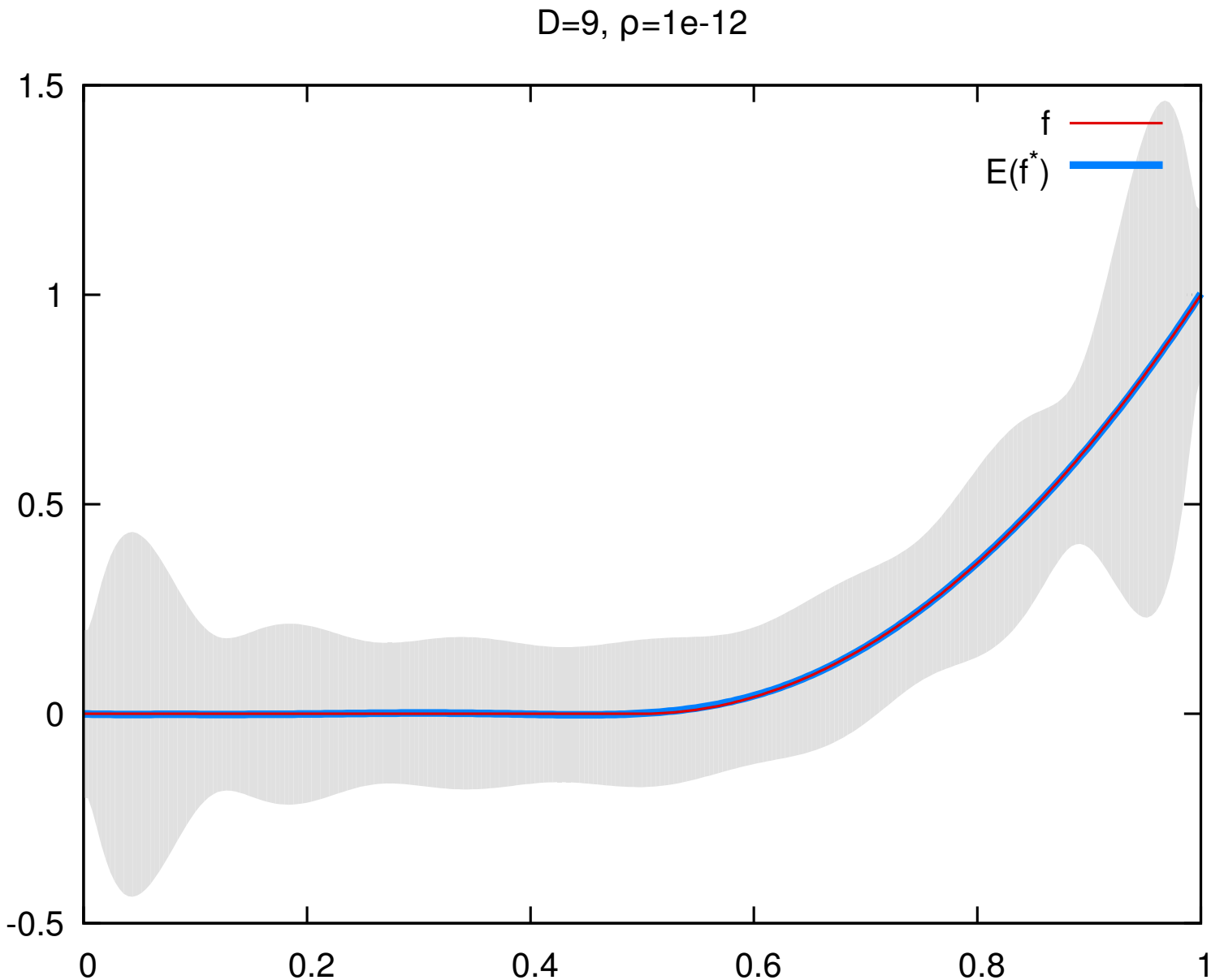

D=9, ρ=1e-7
1.5
1
0.5
0
-0.5
0
0.2
0.4
0.6
0.8
1
f
E(f*)

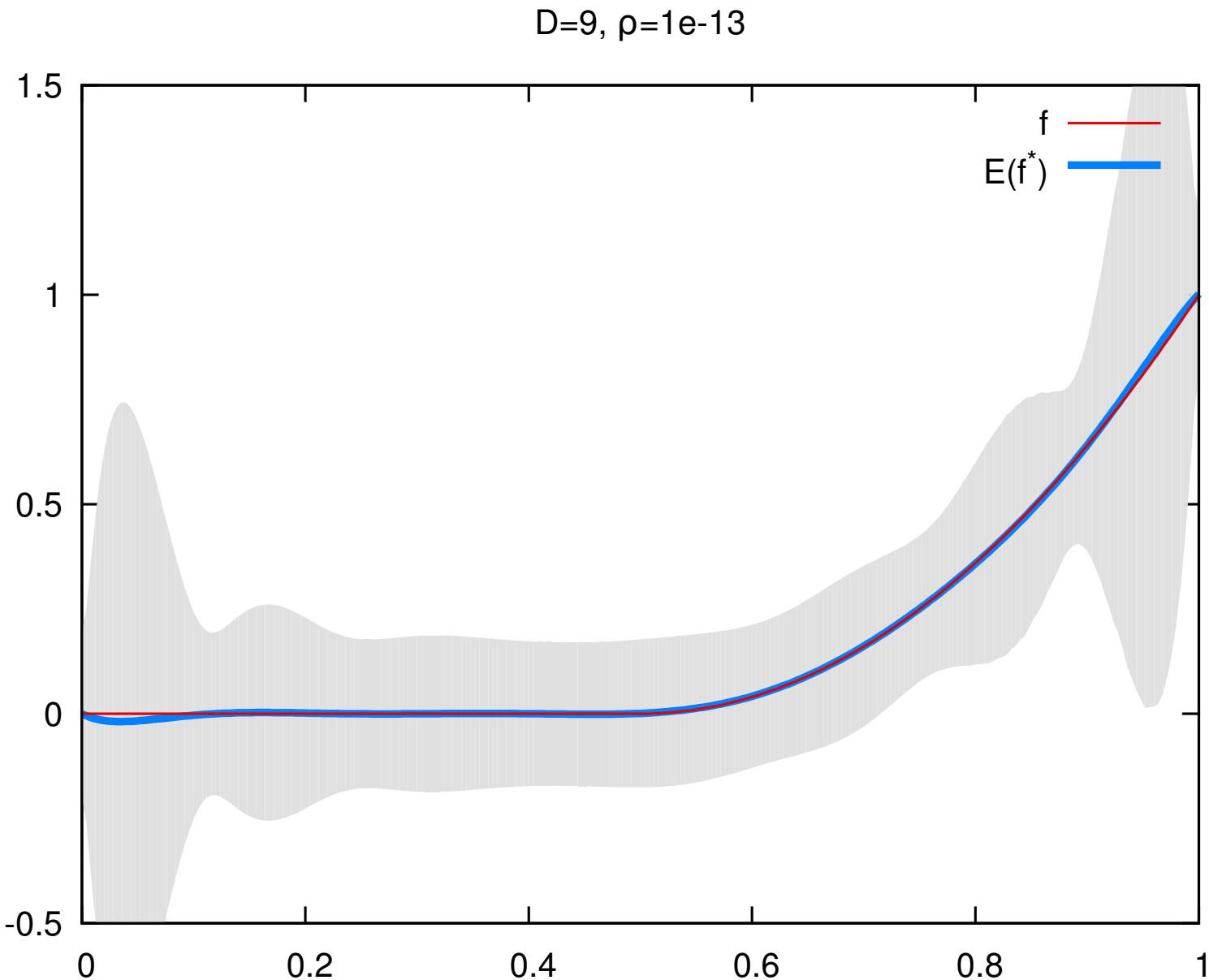

D=9, ρ=1e-8
f
E(f*)
1.5
1
0.5
0
-0.5
0
0.2
0.4
0.6
0.8
1

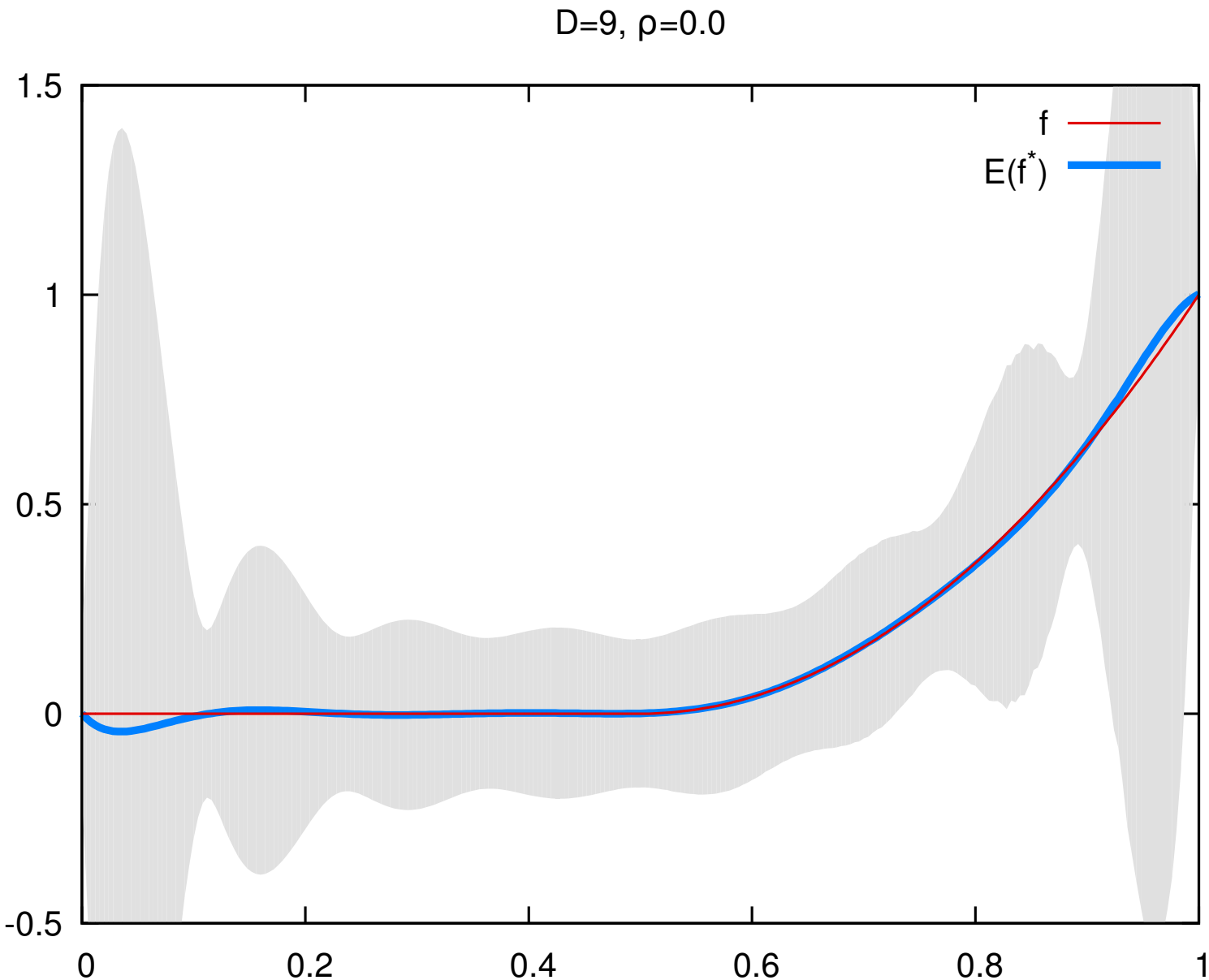

D=9, ρ=1e-9
f
E(f*)
1.5
1
0.5
0
-0.5
0
0.2
0.4
0.6
0.8
1


D=9, ρ=1e-10
f
E(f*)
1.5
1
0.5
0
-0.5
0
0.2
0.4
0.6
0.8
1


D=9, ρ=1e-11
f
E(f*)
1.5
1
0.5
0
-0.5
0
0.2
0.4
0.6
0.8
1


D=9, ρ=1e-12
f
E(f*)
1.5
1
0.5
0
-0.5
0
0.2
0.4
0.6
0.8
1


D=9, ρ=1e-13
f
E(f*)
1.5
1
0.5
0
-0.5
0
0.2
0.4
0.6
0.8
1


D=9, ρ=0.0
1.5
1
0.5
0
-0.5
0
0.2
0.4
0.6
0.8
1
f
E(f*)

Let x be fixed, and $y = f(x)$ the "true" value associated to it.

Let $Y = f^*(x)$ be the value we predict.

If we consider that the training set $\mathscr{D}$ is a random quantity, so is f^*, and consequently Y.

We have

$$\mathbb{E}_{\mathscr{D}}((Y - y)^2)$$

We have

$$\mathbb{E}_{\mathscr{D}}((Y-y)^2) = \mathbb{E}_{\mathscr{D}}(Y^2 - 2Yy + y^2)$$

We have

$$\begin{aligned}\mathbb{E}_{\mathscr{D}}((Y-y)^2) &= \mathbb{E}_{\mathscr{D}}(Y^2 - 2Yy + y^2) \\ &= \mathbb{E}_{\mathscr{D}}(Y^2) - 2\mathbb{E}_{\mathscr{D}}(Y)y + y^2\end{aligned}$$

We have

$$\begin{aligned}
\mathbb{E}_{\mathscr{D}}((Y-y)^2) &= \mathbb{E}_{\mathscr{D}}(Y^2 - 2Yy + y^2) \\
&= \mathbb{E}_{\mathscr{D}}(Y^2) - 2\mathbb{E}_{\mathscr{D}}(Y)y + y^2 \\
&= \underbrace{\mathbb{E}_{\mathscr{D}}(Y^2) - \mathbb{E}_{\mathscr{D}}(Y)^2}_{\mathbb{V}_{\mathscr{D}}(Y)} + \underbrace{\mathbb{E}_{\mathscr{D}}(Y)^2 - 2\mathbb{E}_{\mathscr{D}}(Y)y + y^2}_{(\mathbb{E}_{\mathscr{D}}(Y)-y)^2}
\end{aligned}$$

We have

$$\begin{aligned}
\mathbb{E}_{\mathscr{D}}((Y-y)^2) &= \mathbb{E}_{\mathscr{D}}(Y^2 - 2Yy + y^2) \\
&= \mathbb{E}_{\mathscr{D}}(Y^2) - 2\mathbb{E}_{\mathscr{D}}(Y)y + y^2 \\
&= \underbrace{\mathbb{E}_{\mathscr{D}}(Y^2) - \mathbb{E}_{\mathscr{D}}(Y)^2}_{\mathbb{V}_{\mathscr{D}}(Y)} + \underbrace{\mathbb{E}_{\mathscr{D}}(Y)^2 - 2\mathbb{E}_{\mathscr{D}}(Y)y + y^2}_{(\mathbb{E}_{\mathscr{D}}(Y)-y)^2} \\
&= \underbrace{(\mathbb{E}_{\mathscr{D}}(Y) - y)^2}_{\text{Bias}} + \underbrace{\mathbb{V}_{\mathscr{D}}(Y)}_{\text{Variance}}
\end{aligned}$$

We have

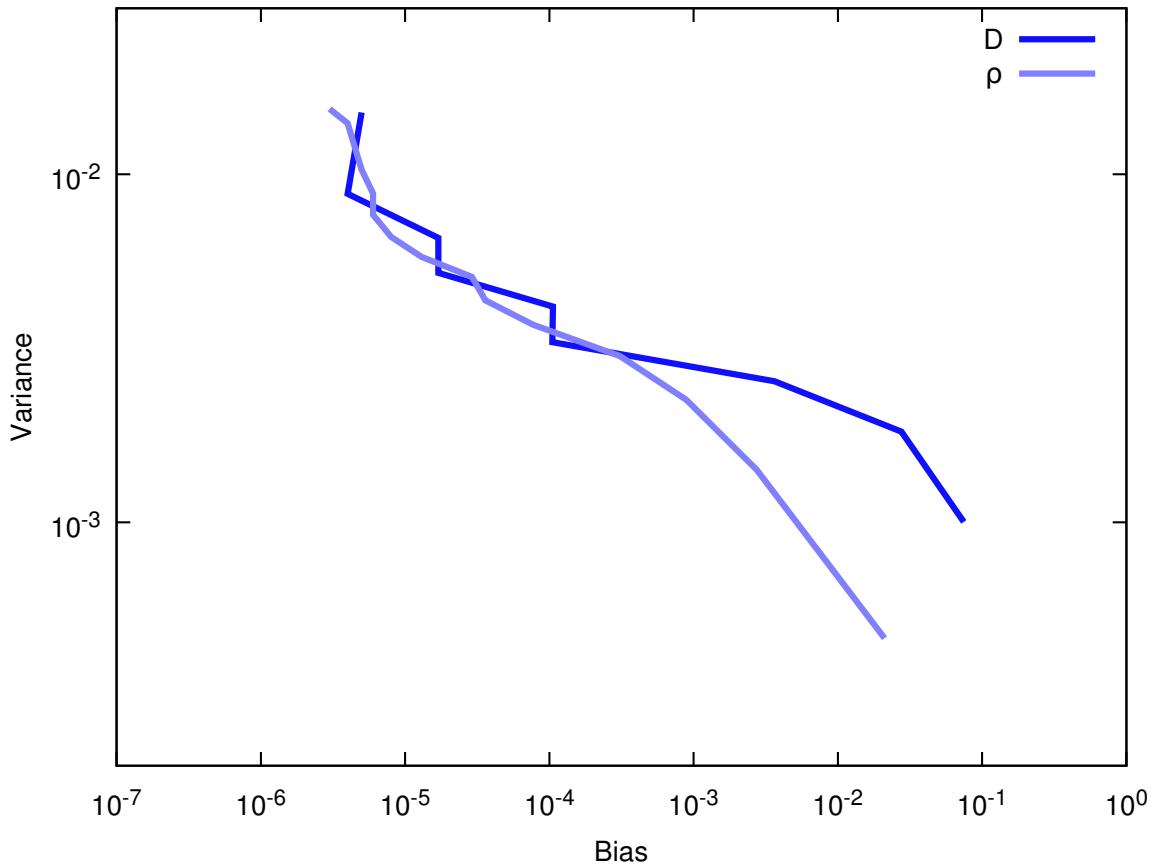
$$\begin{aligned}
\mathbb{E}_{\mathscr{D}}((Y-y)^2) &= \mathbb{E}_{\mathscr{D}}(Y^2 - 2Yy + y^2) \\
&= \mathbb{E}_{\mathscr{D}}(Y^2) - 2\mathbb{E}_{\mathscr{D}}(Y)y + y^2 \\
&= \underbrace{\mathbb{E}_{\mathscr{D}}(Y^2) - \mathbb{E}_{\mathscr{D}}(Y)^2}_{\mathbb{V}_{\mathscr{D}}(Y)} + \underbrace{\mathbb{E}_{\mathscr{D}}(Y)^2 - 2\mathbb{E}_{\mathscr{D}}(Y)y + y^2}_{(\mathbb{E}_{\mathscr{D}}(Y)-y)^2} \\
&= \underbrace{(\mathbb{E}_{\mathscr{D}}(Y) - y)^2}_{\text{Bias}} + \underbrace{\mathbb{V}_{\mathscr{D}}(Y)}_{\text{Variance}}
\end{aligned}$$

This is the bias-variance decomposition:

- the bias term quantifies how much the model fits the data on average,
- the variance term quantifies how much the model changes across data-sets.

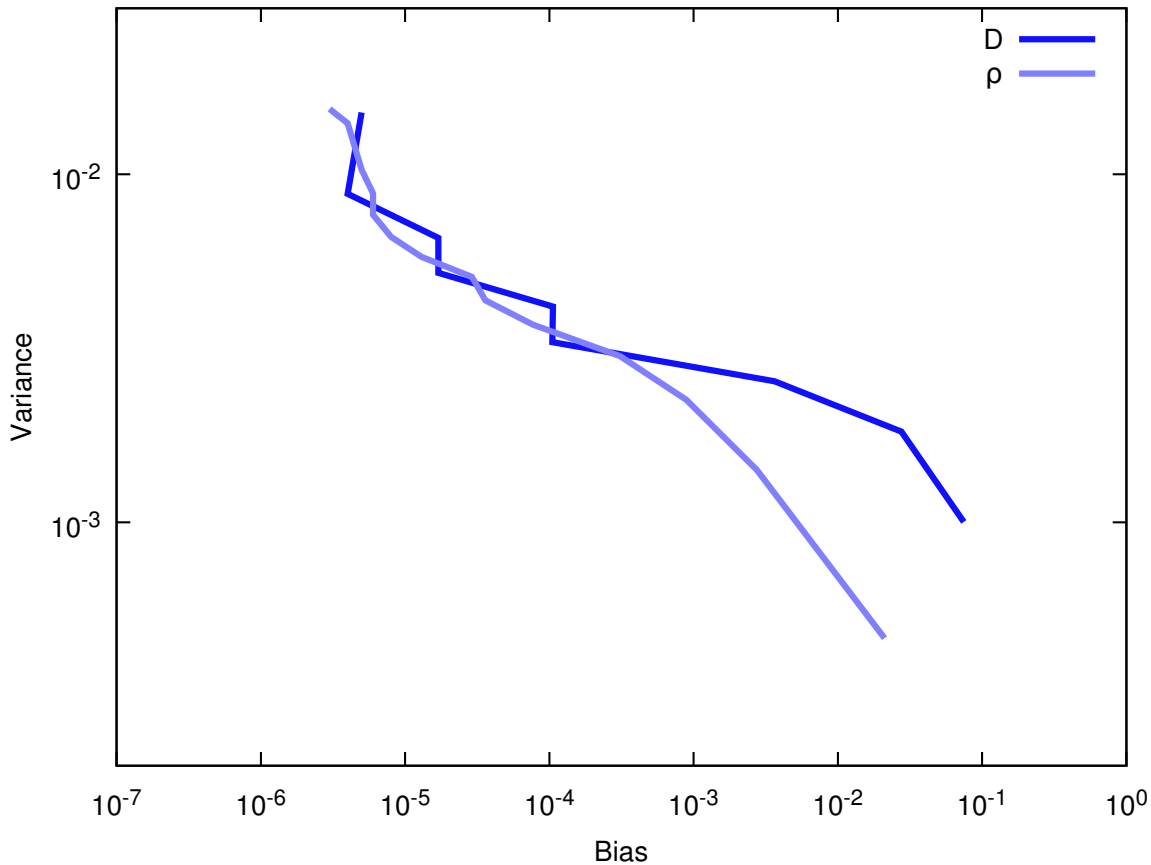
(Geman and Bienenstock, 1992)

From this comes the **bias variance tradeoff:**



Reducing the capacity makes f^* fit the data less on average, which increases the bias term.

From this comes the **bias variance tradeoff:**



Reducing the capacity makes f^* fit the data less on average, which increases the bias term. Increasing the capacity makes f^* vary a lot with the training data, which increases the variance term.

Is all this probabilistic?

Conceptually model-fitting and regularization can be interpreted as Bayesian inference.

Conceptually model-fitting and regularization can be interpreted as Bayesian inference.

This approach consists of **modeling the parameters A of the model themselves as random quantities with a prior μ_A.**

Conceptually model-fitting and regularization can be interpreted as Bayesian inference.

This approach consists of **modeling the parameters A of the model themselves as random quantities with a prior μ_A.**

By looking at the data $\mathscr{D}$, we can estimate a posterior distribution for the said parameters,

$$\mu_A(\alpha \mid \mathscr{D} = \mathbf{d}) \propto \mu_{\mathscr{D}}(\mathbf{d} \mid A = \alpha)\, \mu_A(\alpha),$$

and from that their most likely values.

Conceptually model-fitting and regularization can be interpreted as Bayesian inference.

This approach consists of **modeling the parameters A of the model themselves as random quantities with a prior μ_A.**

By looking at the data $\mathscr{D}$, we can estimate a posterior distribution for the said parameters,

$$\mu_A(\alpha \mid \mathscr{D} = \mathbf{d}) \propto \mu_{\mathscr{D}}(\mathbf{d} \mid A = \alpha)\, \mu_A(\alpha),$$

and from that their most likely values.

So instead of a penalty term, we define a prior distribution, which is usually more intellectually satisfying.

For instance, consider the model

$$\forall n,\ Y_n = \sum_{d=0}^{D} A_d\, X_n^d + \Delta_n,$$

where

$$\forall d,\ A_d \sim \mathcal{N}(0, \xi),\ \forall n,\ X_n \sim \mu_X,\ \Delta_n \sim \mathcal{N}\,(0, \sigma)$$

all independent.

For instance, consider the model

$$\forall n,\ Y_n = \sum_{d=0}^{D} A_d\, X_n^d + \Delta_n,$$

where

$$\forall d,\ A_d \sim \mathcal{N}(0, \xi),\ \forall n,\ X_n \sim \mu_X,\ \Delta_n \sim \mathcal{N}(0, \sigma)$$

all independent.

For clarity, let $A = (A_0, \ldots, A_D)$ and $\alpha = (\alpha_0, \ldots, \alpha_D)$.

Remember that $\mathscr{D} = \{(X_1, Y_1), \ldots, (X_N, Y_N)\}$ is the (random) training set and $\mathbf{d} = \{(x_1, y_1), \ldots, (x_N, y_N)\}$ is a realization.

$$\log \mu_A(\alpha \mid \mathscr{D} = \mathbf{d})$$

$$\log \mu_A(\alpha \mid \mathscr{D} = \mathbf{d})$$

$$= \log \frac{\mu_{\mathscr{D}}(\mathbf{d} \mid A = \alpha)\, \mu_A(\alpha)}{\mu_{\mathscr{D}}(\mathbf{d})}$$

$$
\begin{aligned}
&\log \mu_A(\alpha \mid \mathscr{D} = \mathbf{d}) \\
&\qquad = \log \frac{\mu_{\mathscr{D}}(\mathbf{d} \mid A = \alpha)\, \mu_A(\alpha)}{\mu_{\mathscr{D}}(\mathbf{d})} \\
&\qquad = \log \mu_{\mathscr{D}}(\mathbf{d} \mid A = \alpha) + \log \mu_A(\alpha) - \log Z
\end{aligned}
$$

$$
\begin{aligned}
&\log \mu_A(\alpha \mid \mathscr{D} = \mathbf{d}) \\
&\qquad = \log \frac{\mu_{\mathscr{D}}(\mathbf{d} \mid A = \alpha)\, \mu_A(\alpha)}{\mu_{\mathscr{D}}(\mathbf{d})} \\
&\qquad = \log \mu_{\mathscr{D}}(\mathbf{d} \mid A = \alpha) + \log \mu_A(\alpha) - \log Z \\
&\qquad = \log \prod_n \mu(x_n, y_n \mid A = \alpha) + \log \mu_A(\alpha) - \log Z
\end{aligned}
$$

$$
\begin{aligned}
&\log \mu_A(\alpha \mid \mathscr{D} = \mathbf{d}) \\
&\qquad = \log \frac{\mu_{\mathscr{D}}(\mathbf{d} \mid A = \alpha)\, \mu_A(\alpha)}{\mu_{\mathscr{D}}(\mathbf{d})} \\
&\qquad = \log \mu_{\mathscr{D}}(\mathbf{d} \mid A = \alpha) + \log \mu_A(\alpha) - \log Z \\
&\qquad = \log \prod_n \mu(x_n, y_n \mid A = \alpha) + \log \mu_A(\alpha) - \log Z \\
&\qquad = \log \prod_n \mu(y_n \mid X_n = x_n, A = \alpha)\, \underbrace{\mu(x_n \mid A = \alpha)}_{= \mu(x_n)} + \log \mu_A(\alpha) - \log Z
\end{aligned}
$$

$$
\begin{aligned}
&\log \mu_A(\alpha \mid \mathscr{D} = \mathbf{d}) \\
&\qquad = \log \frac{\mu_{\mathscr{D}}(\mathbf{d} \mid A = \alpha)\, \mu_A(\alpha)}{\mu_{\mathscr{D}}(\mathbf{d})} \\
&\qquad = \log \mu_{\mathscr{D}}(\mathbf{d} \mid A = \alpha) + \log \mu_A(\alpha) - \log Z \\
&\qquad = \log \prod_n \mu(x_n, y_n \mid A = \alpha) + \log \mu_A(\alpha) - \log Z \\
&\qquad = \log \prod_n \mu(y_n \mid X_n = x_n, A = \alpha)\, \underbrace{\mu(x_n \mid A = \alpha)}_{= \mu(x_n)} + \log \mu_A(\alpha) - \log Z \\
&\qquad = \log \prod_n \mu(y_n \mid X_n = x_n, A = \alpha) + \log \mu_A(\alpha) - \log Z'
\end{aligned}
$$

$$
\begin{aligned}
&\log \mu_A(\alpha \mid \mathscr{D} = \mathbf{d}) \\
&\qquad = \log \frac{\mu_{\mathscr{D}}(\mathbf{d} \mid A = \alpha)\, \mu_A(\alpha)}{\mu_{\mathscr{D}}(\mathbf{d})} \\
&\qquad = \log \mu_{\mathscr{D}}(\mathbf{d} \mid A = \alpha) + \log \mu_A(\alpha) - \log Z \\
&\qquad = \log \prod_n \mu(x_n, y_n \mid A = \alpha) + \log \mu_A(\alpha) - \log Z \\
&\qquad = \log \prod_n \mu(y_n \mid X_n = x_n, A = \alpha)\, \underbrace{\mu(x_n \mid A = \alpha)}_{= \mu(x_n)} + \log \mu_A(\alpha) - \log Z \\
&\qquad = \log \prod_n \mu(y_n \mid X_n = x_n, A = \alpha) + \log \mu_A(\alpha) - \log Z' \\
&\qquad = \underbrace{-\frac{1}{2\sigma^2} \sum_n \left(y_n - \sum_d \alpha_d x_n^d \right)^2}_{\text{Gaussian noise on } Y\text{s}} - \underbrace{\frac{1}{2\xi^2} \sum_d \alpha_d^2}_{\text{Gaussian prior on } A\text{s}} - \log Z''.
\end{aligned}
$$

Taking $\rho = \sigma^2/\xi^2$ gives the penalty term of the previous slides.

$$
\begin{aligned}
&\log \mu_A(\alpha \mid \mathscr{D} = \mathbf{d}) \\
&\qquad = \log \frac{\mu_{\mathscr{D}}(\mathbf{d} \mid A = \alpha)\, \mu_A(\alpha)}{\mu_{\mathscr{D}}(\mathbf{d})} \\
&\qquad = \log \mu_{\mathscr{D}}(\mathbf{d} \mid A = \alpha) + \log \mu_A(\alpha) - \log Z \\
&\qquad = \log \prod_n \mu(x_n, y_n \mid A = \alpha) + \log \mu_A(\alpha) - \log Z \\
&\qquad = \log \prod_n \mu(y_n \mid X_n = x_n, A = \alpha)\, \underbrace{\mu(x_n \mid A = \alpha)}_{= \mu(x_n)} + \log \mu_A(\alpha) - \log Z \\
&\qquad = \log \prod_n \mu(y_n \mid X_n = x_n, A = \alpha) + \log \mu_A(\alpha) - \log Z' \\
&\qquad = \underbrace{-\frac{1}{2\sigma^2} \sum_n \left(y_n - \sum_d \alpha_d x_n^d \right)^2}_{\text{Gaussian noise on } Y\text{s}} - \underbrace{\frac{1}{2\xi^2} \sum_d \alpha_d^2}_{\text{Gaussian prior on } A\text{s}} - \log Z''.
\end{aligned}
$$

Taking $\rho = \sigma^2/\xi^2$ gives the penalty term of the previous slides.

Regularization seen through that prism is intuitive: The stronger the prior, the more evidence you need to deviate from it.

Proper evaluation protocols

Learning algorithms, in particular deep-learning ones, require the tuning of many meta-parameters.

Learning algorithms, in particular deep-learning ones, require the tuning of many meta-parameters.

These parameters have a strong impact on the performance, resulting in a "meta" over-fitting through experiments.

Learning algorithms, in particular deep-learning ones, require the tuning of many meta-parameters.

These parameters have a strong impact on the performance, resulting in a "meta" over-fitting through experiments.

We must be extra careful with performance estimation.

Learning algorithms, in particular deep-learning ones, require the tuning of many meta-parameters.

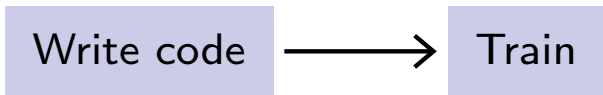
These parameters have a strong impact on the performance, resulting in a "meta" over-fitting through experiments.

We must be extra careful with performance estimation.

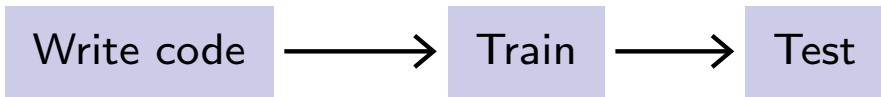
Running 100 times the same experiment on MNIST, with randomized weights, we get:

Worst	Median	Best
1.3%	1.0%	0.82%

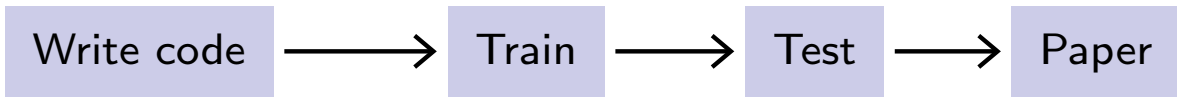
The ideal development cycle is



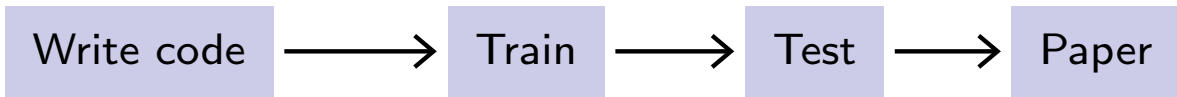
The ideal development cycle is



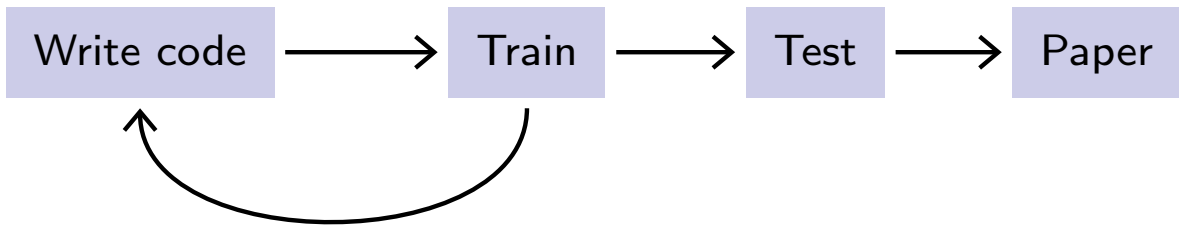
The ideal development cycle is



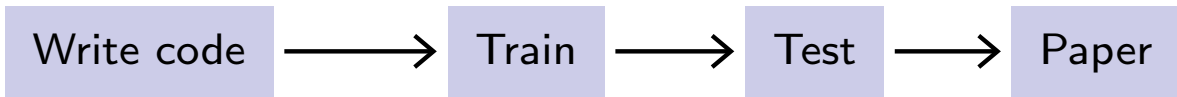
The ideal development cycle is



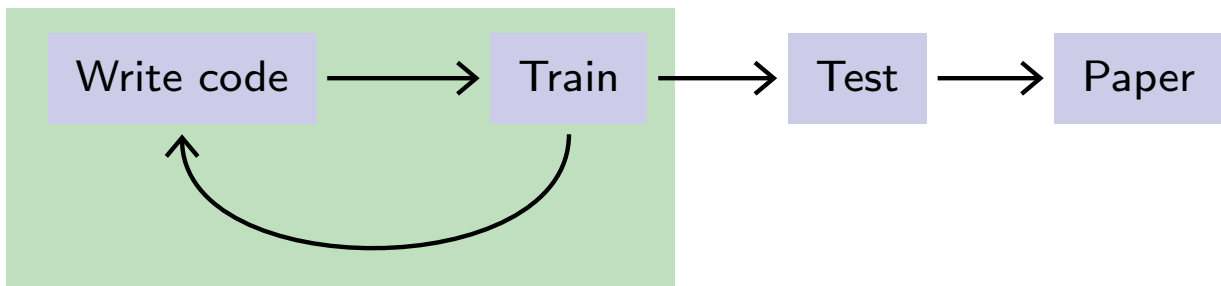
or in practice something like



The ideal development cycle is

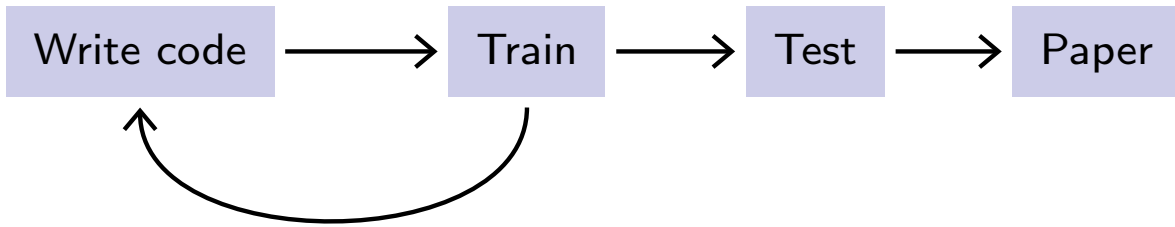


or in practice something like

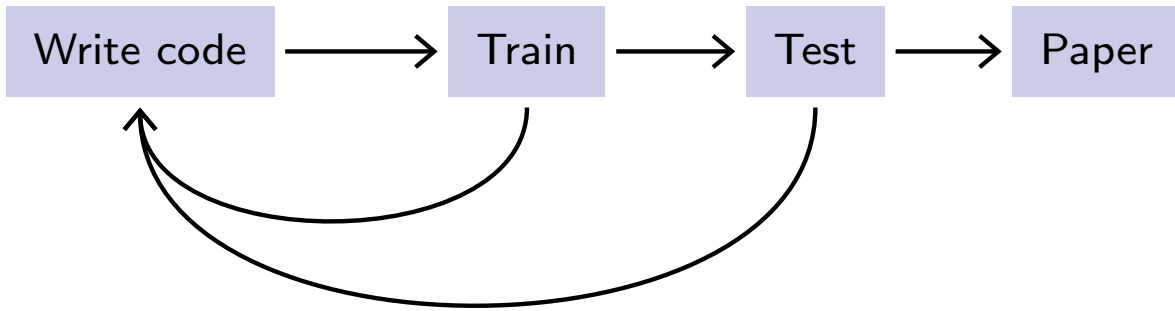


There may be over-fitting, but it does not bias the final performance evaluation.

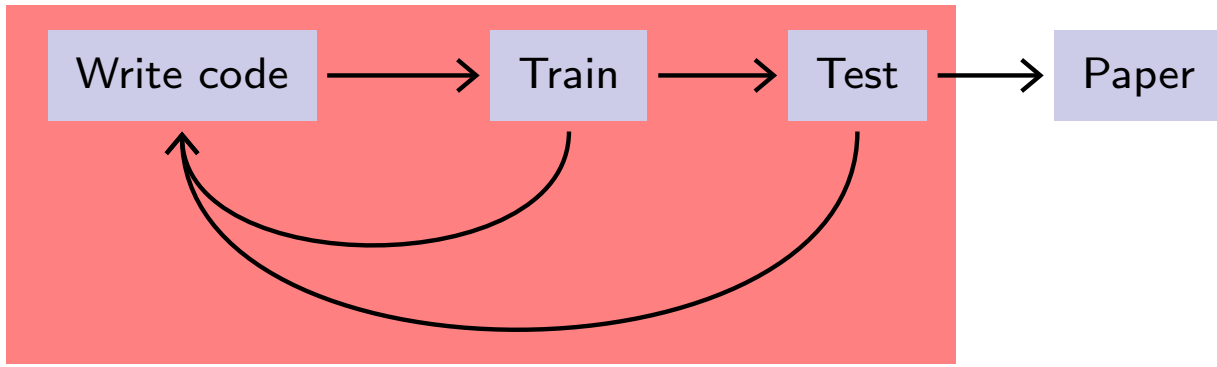
Unfortunately, it often looks like



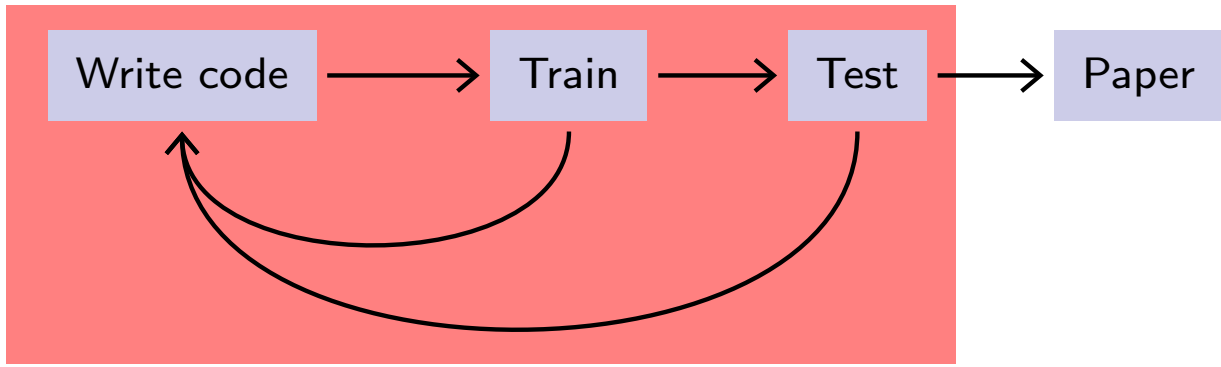
Unfortunately, it often looks like



Unfortunately, it often looks like

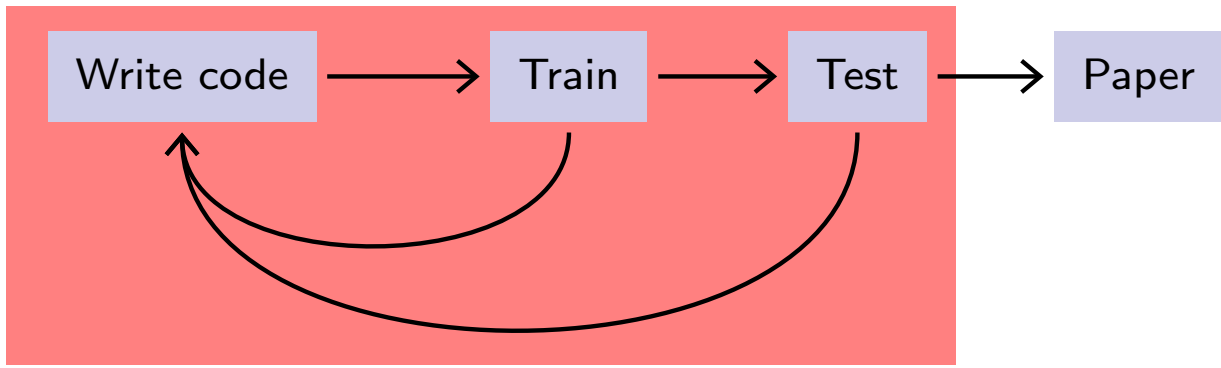


Unfortunately, it often looks like

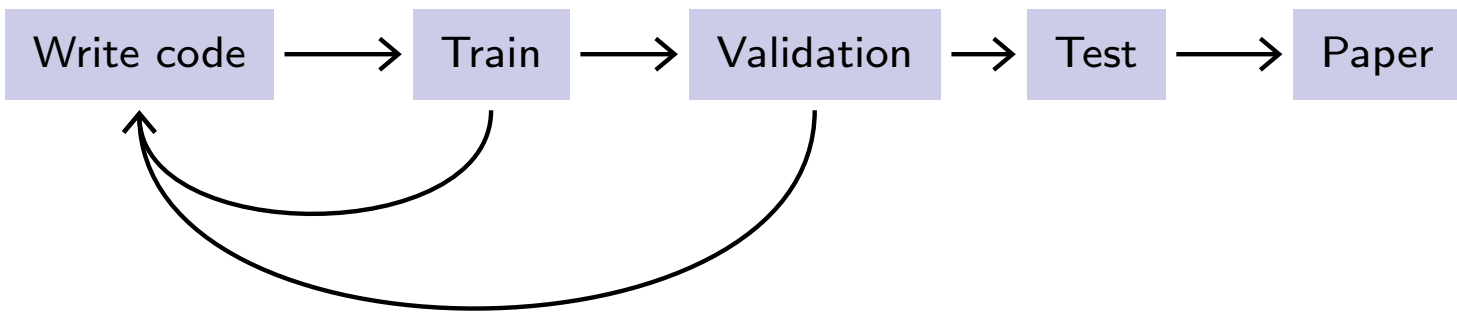


This should be avoided at all costs. The standard strategy is to have a separate validation set for the tuning.

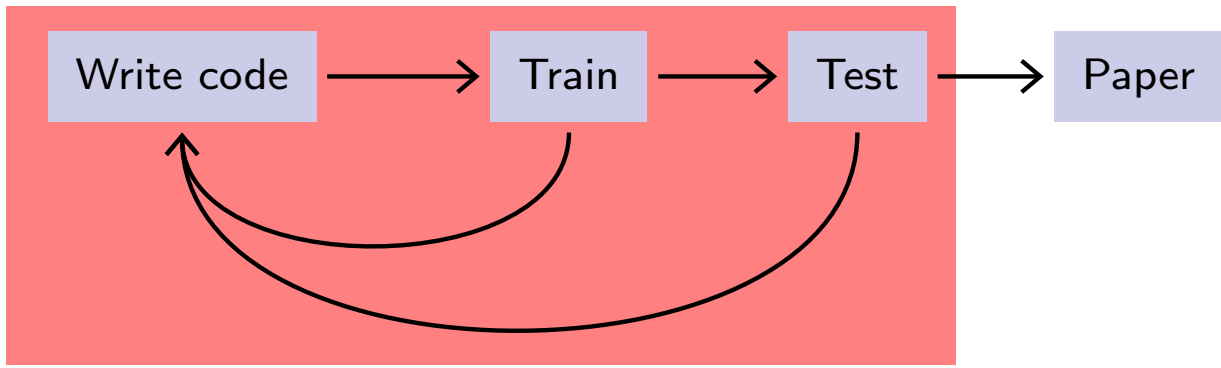
Unfortunately, it often looks like



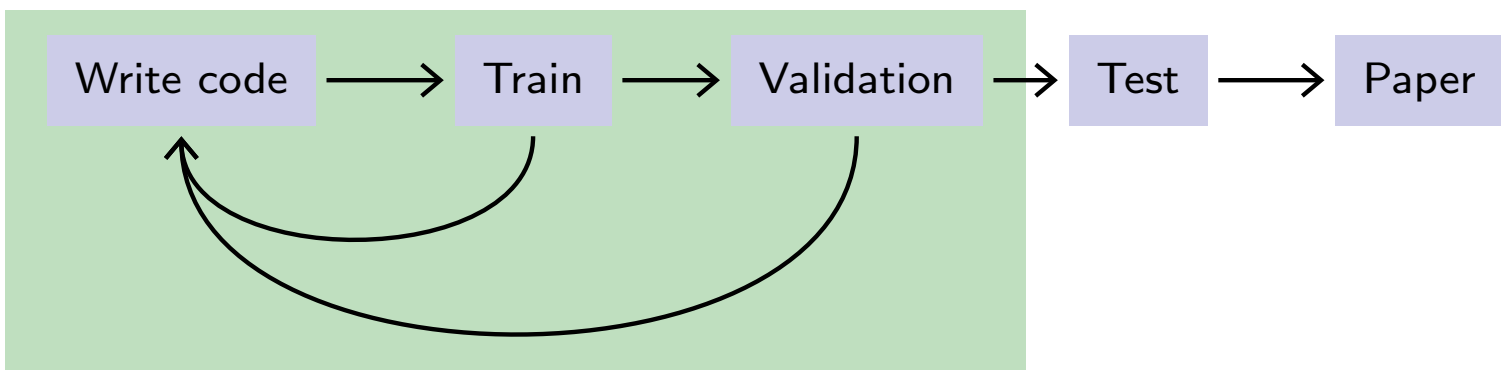
This should be avoided at all costs. The standard strategy is to have a separate validation set for the tuning.



Unfortunately, it often looks like



This should be avoided at all costs. The standard strategy is to have a separate validation set for the tuning.



When data is scarce, one can use cross-validation: average through multiple random splits of the data in a train and a validation sets.

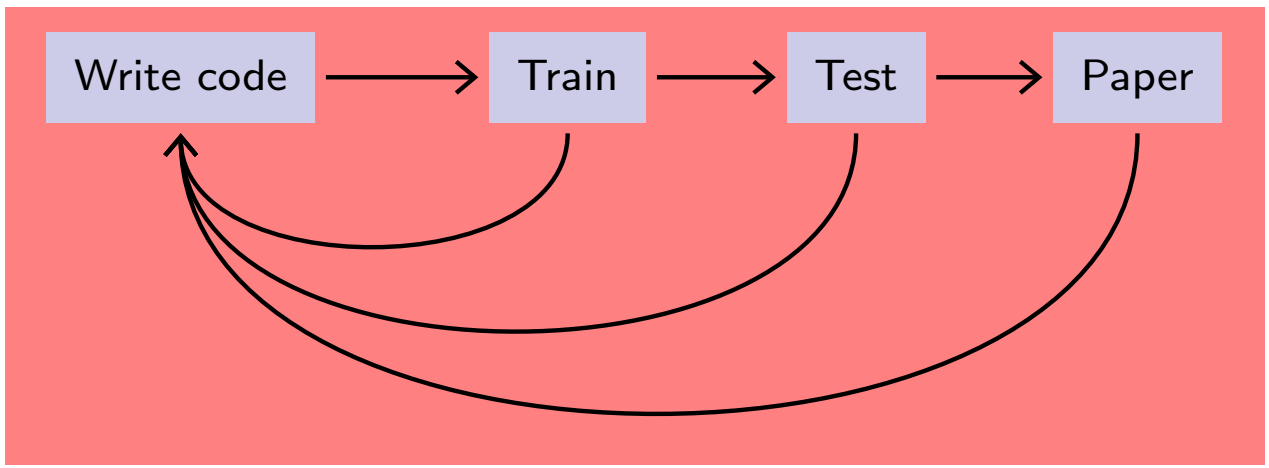
When data is scarce, one can use cross-validation: average through multiple random splits of the data in a train and a validation sets.

There is no unbiased estimator of the variance of cross-validation valid under all distributions (Bengio and Grandvalet, 2004).

Some data-sets (MNIST!) have been used by thousands of researchers, over millions of experiments, in hundreds of papers.

Some data-sets (MNIST!) have been used by thousands of researchers, over millions of experiments, in hundreds of papers.

The global overall process looks more like



"Cheating" in machine learning, from bad to "are you kidding?":

- "Early evaluation stopping",
- meta-parameter (over-)tuning,
- data-set selection,
- algorithm data-set specific clauses,
- seed selection.

"Cheating" in machine learning, from bad to "are you kidding?":

- "Early evaluation stopping",
- meta-parameter (over-)tuning,
- data-set selection,
- algorithm data-set specific clauses,
- seed selection.

Top-tier conference are demanding regarding experiments, and are biased against "complicated" pipelines.

The community pushes toward accessible implementations, reference data-sets, leader boards, and constant upgrades of benchmarks.

Standard clustering and embedding

Deep learning models combine embeddings and dimension reduction operations.

They parametrize and re-parametrize multiple times the input signal under more invariant and interpretable forms.

Deep learning models combine embeddings and dimension reduction operations.

They parametrize and re-parametrize multiple times the input signal under more invariant and interpretable forms.

To get an intuition of how this is possible, we consider here two standard algorithms:

- K-means, and
- Principal Component Analysis (PCA).

Deep learning models combine embeddings and dimension reduction operations.

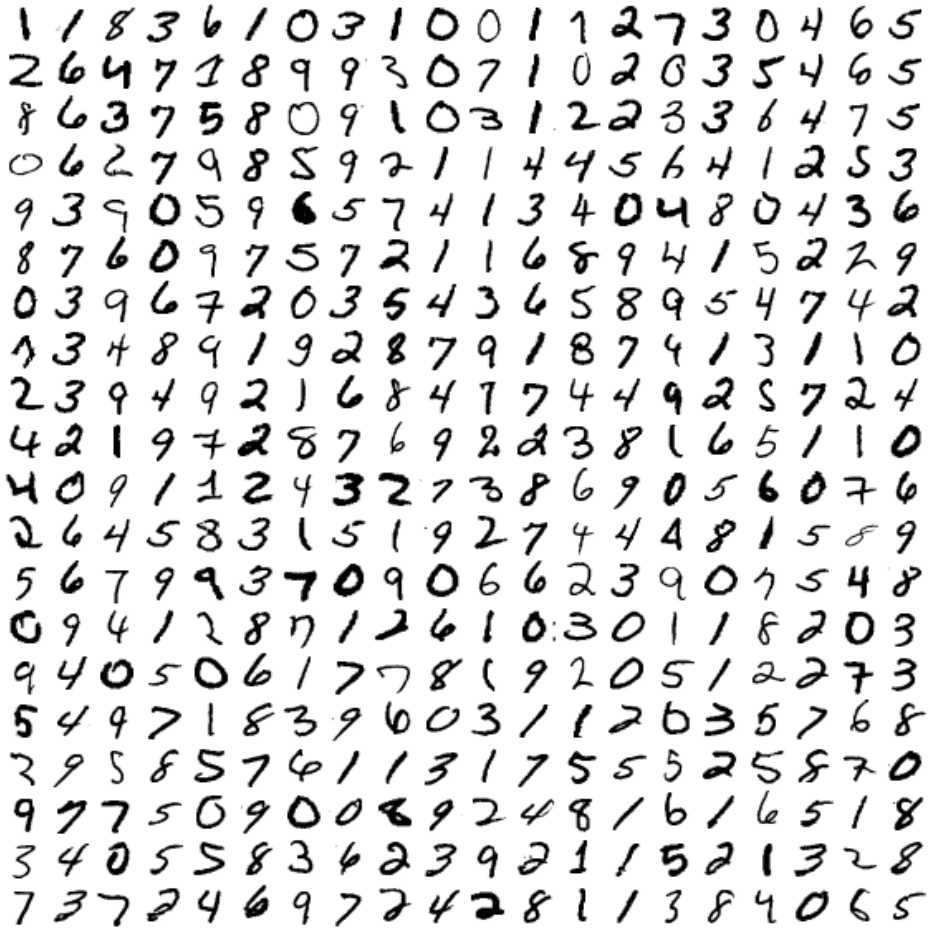
They parametrize and re-parametrize multiple times the input signal under more invariant and interpretable forms.

To get an intuition of how this is possible, we consider here two standard algorithms:

- K-means, and
- Principal Component Analysis (PCA).

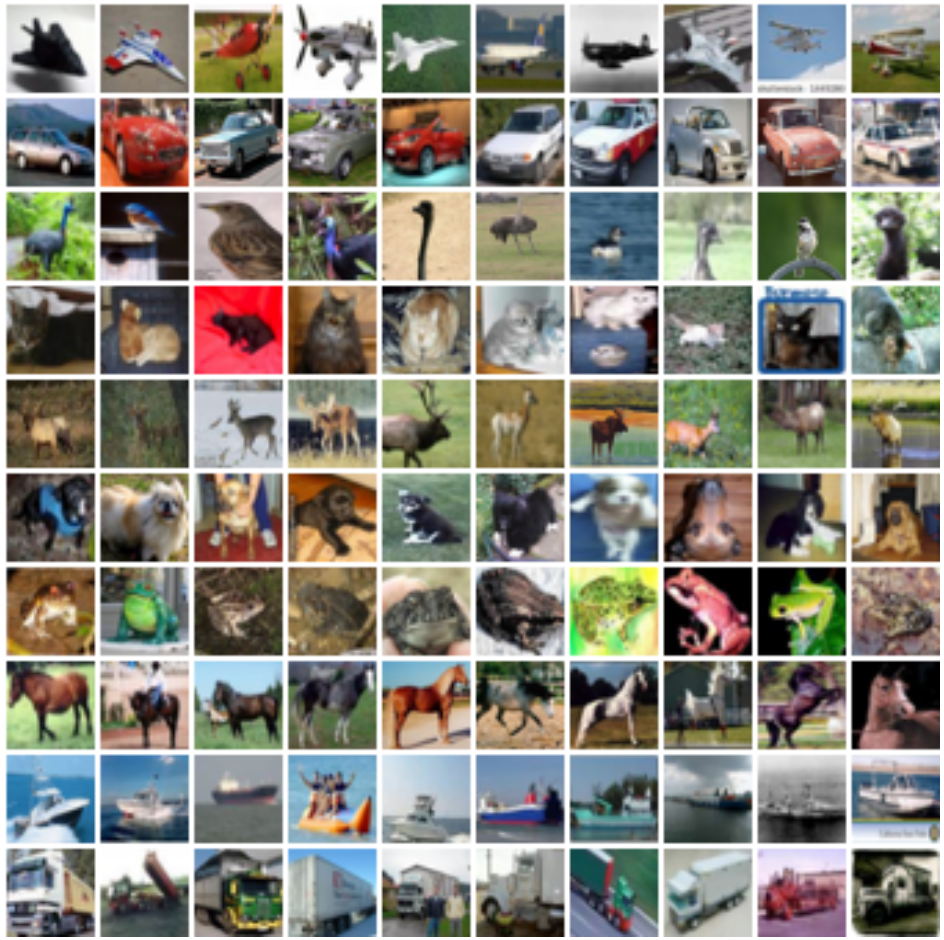
We will illustrate these methods on our two favorite data-sets.

MNIST data-set



28×28 grayscale images, 60k train samples, 10k test samples.

CIFAR10 data-set



32×32 color images, 50k train samples, 10k test samples.

(Krizhevsky, 2009, chap. 3)

Given

$$x_n \in \mathbb{R}^D, \ n = 1, \ldots, N,$$

and a fixed number of clusters $K > 0$, K-means tries to find K "centroids" that span uniformly the training population.

Given

$$x_n \in \mathbb{R}^D, \ n = 1, \ldots, N,$$

and a fixed number of clusters $K > 0$, K-means tries to find K "centroids" that span uniformly the training population.

Given a point, the index of its closest centroid is a good coding.

Formally, [Lloyd's algorithm for] K-means (approximately) solves

$$\operatorname*{argmin}_{c_1,\ldots,c_K \in \mathbb{R}^D} \sum_n \min_k \|x_n - c_k\|^2.$$

Formally, [Lloyd's algorithm for] K-means (approximately) solves

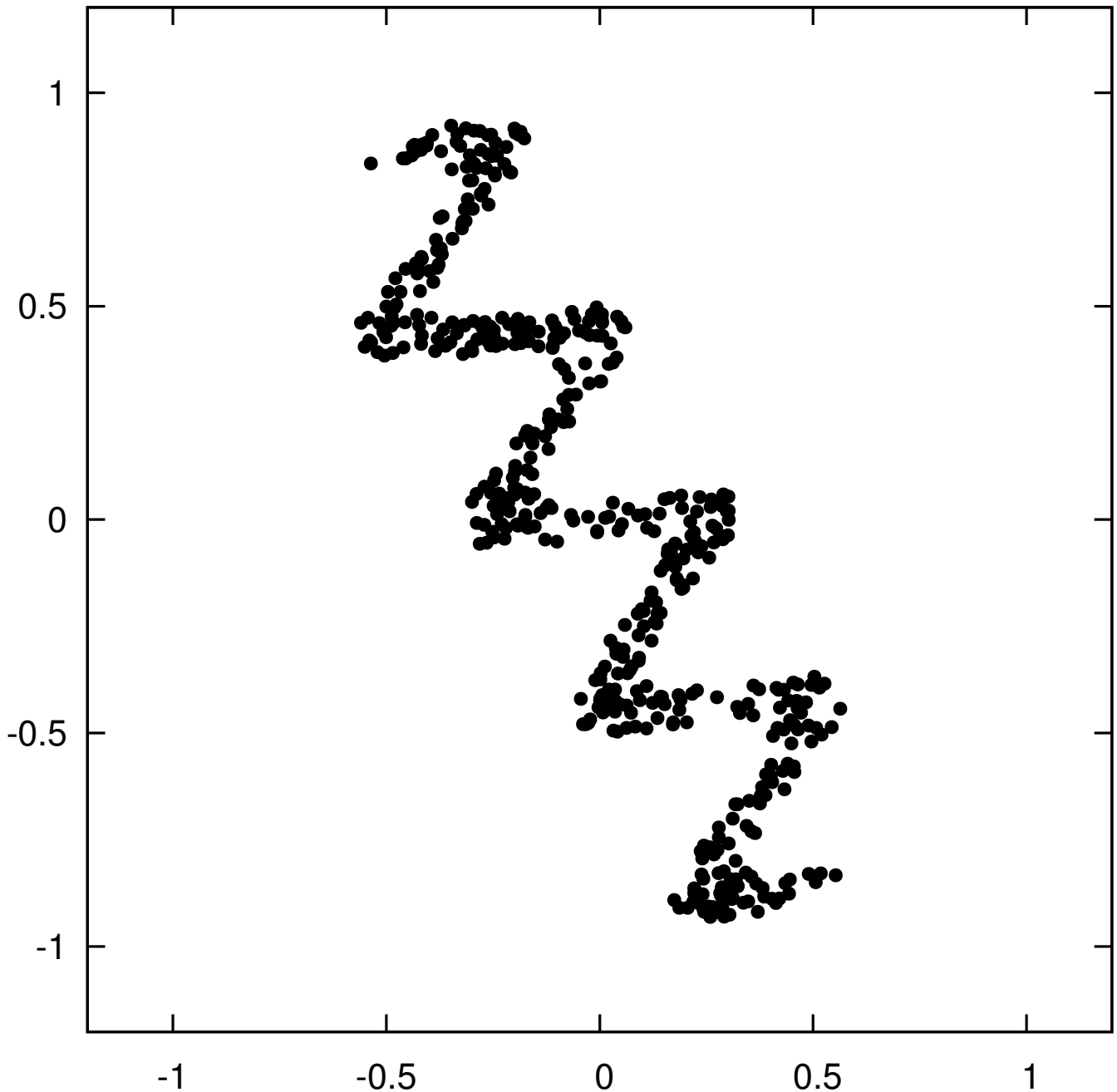
$$\underset{c_1,\ldots,c_K \in \mathbb{R}^D}{\operatorname{argmin}} \sum_n \min_k \|x_n - c_k\|^2.$$

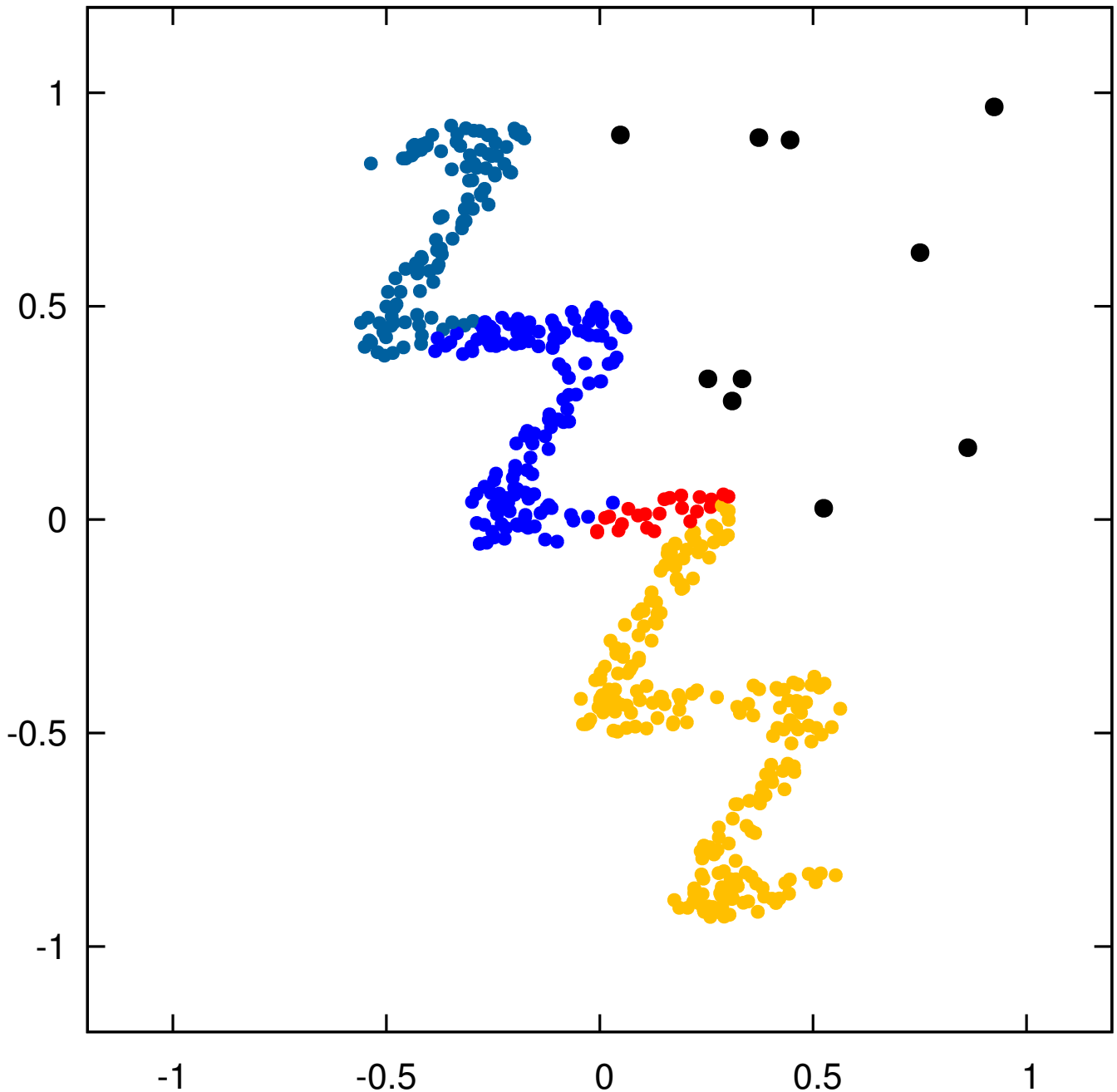
This is achieved with a random initialization of $c_1^0, \ldots c_K^0$ followed by repeating until convergence:

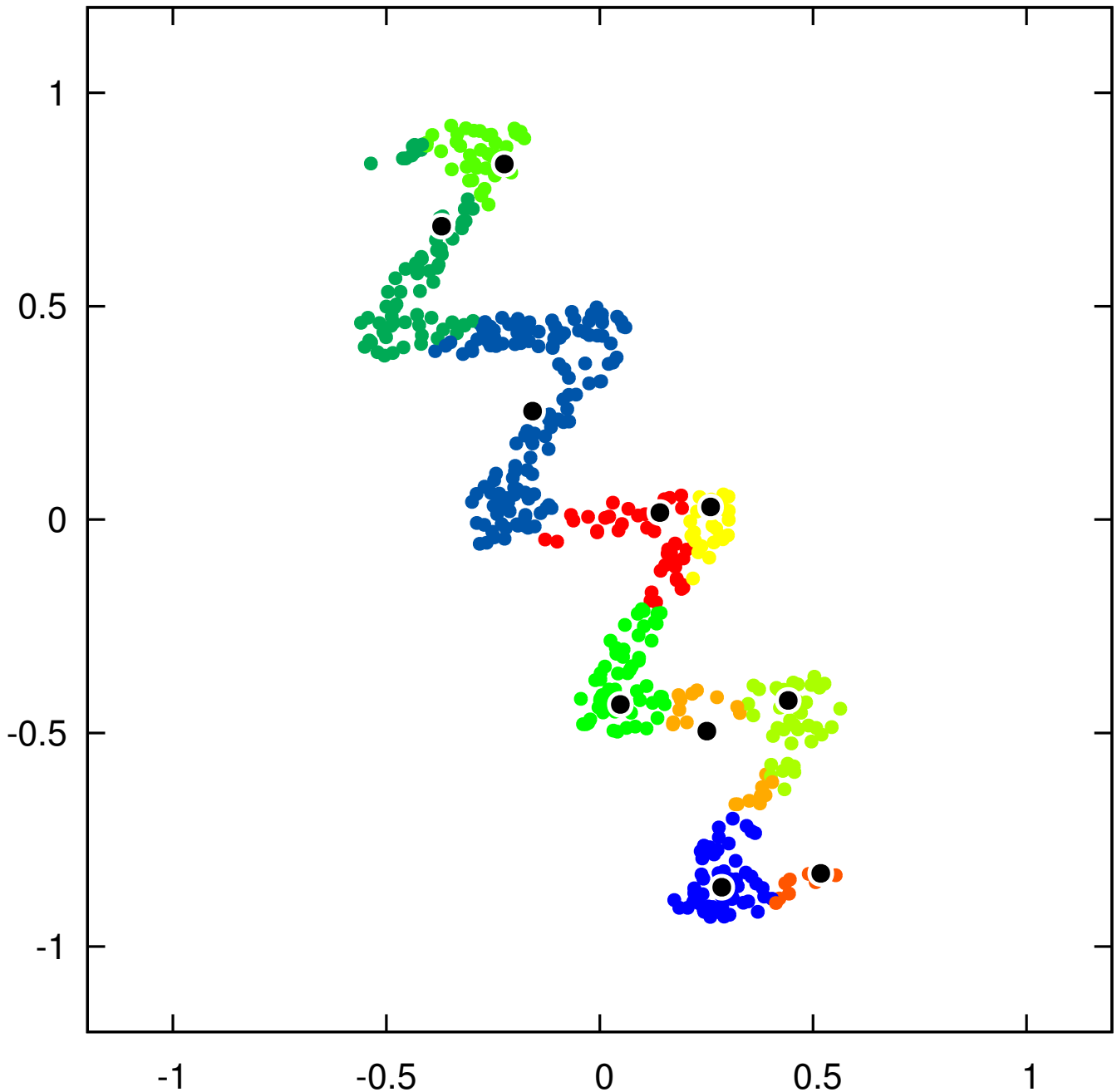
$$\forall n, k_n^t = \underset{k}{\operatorname{argmin}} \|x_n - c_k^t\| \quad (1)$$

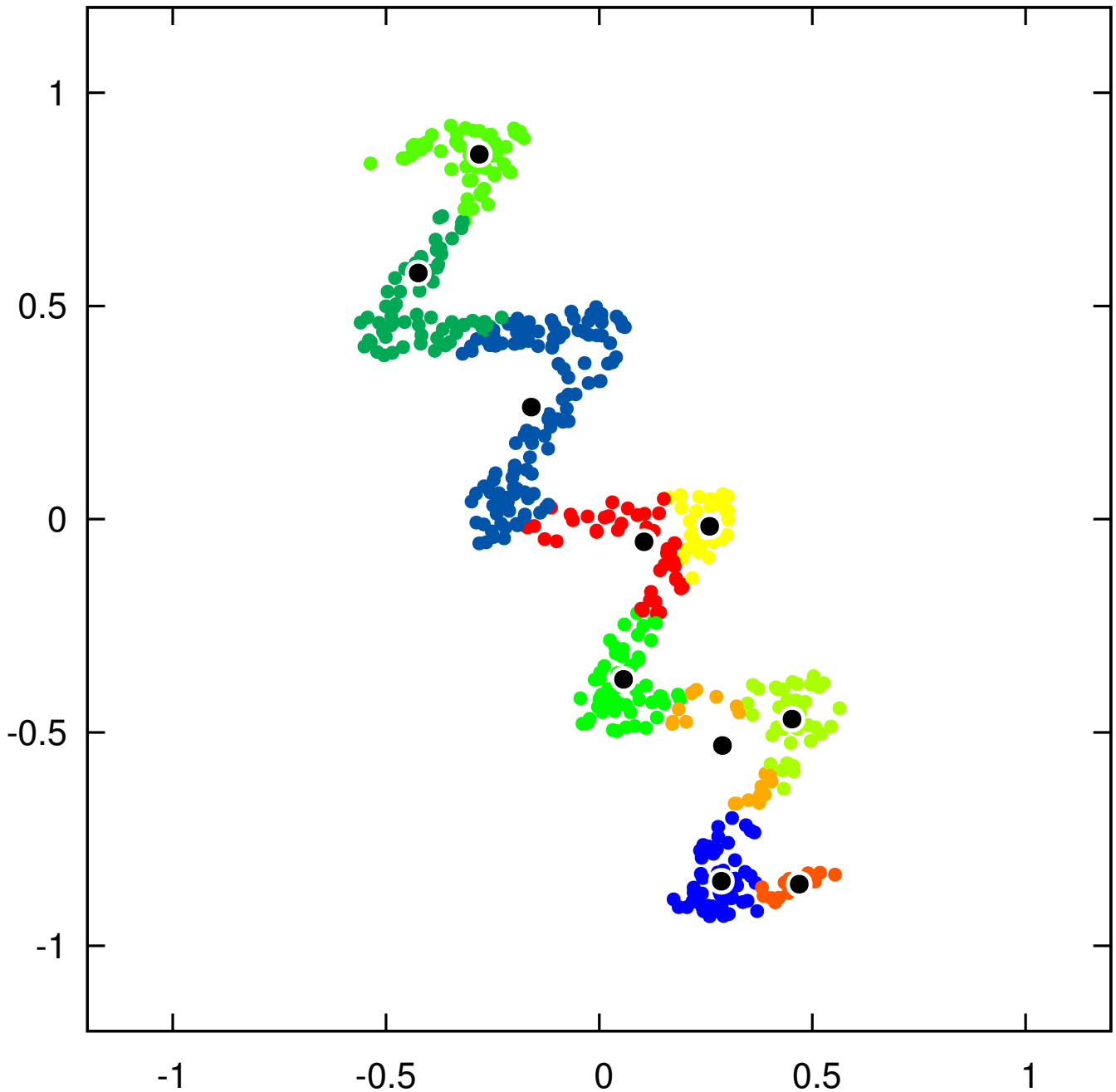
$$\forall k, c_k^{t+1} = \frac{1}{|n : k_n^t = k|} \sum_{n:k_n^t=k} x_n \quad (2)$$

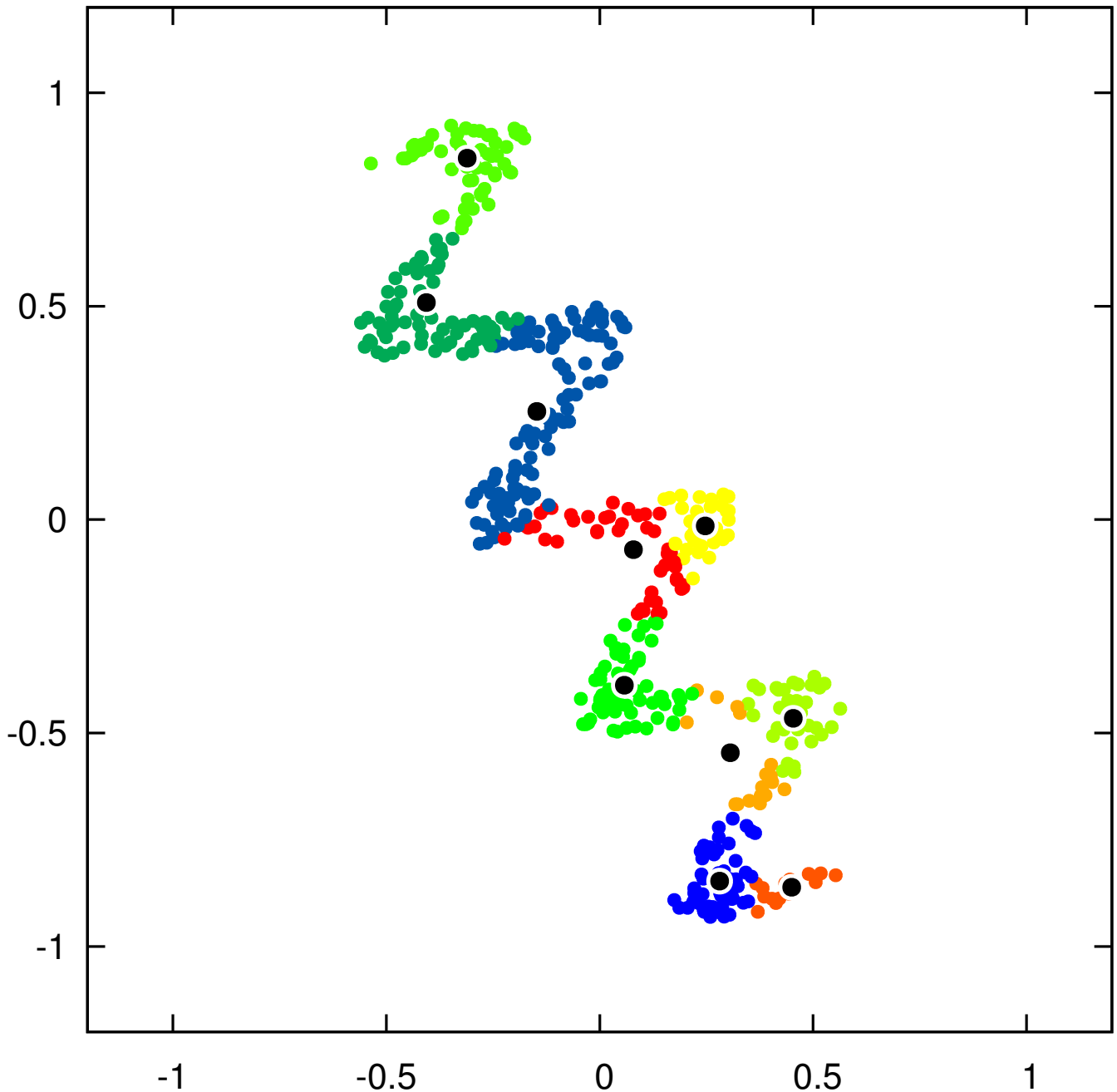
At every iteration, (1) each sample is associated to its closest centroid's cluster, and (2) each centroid is updated to the average of its cluster.

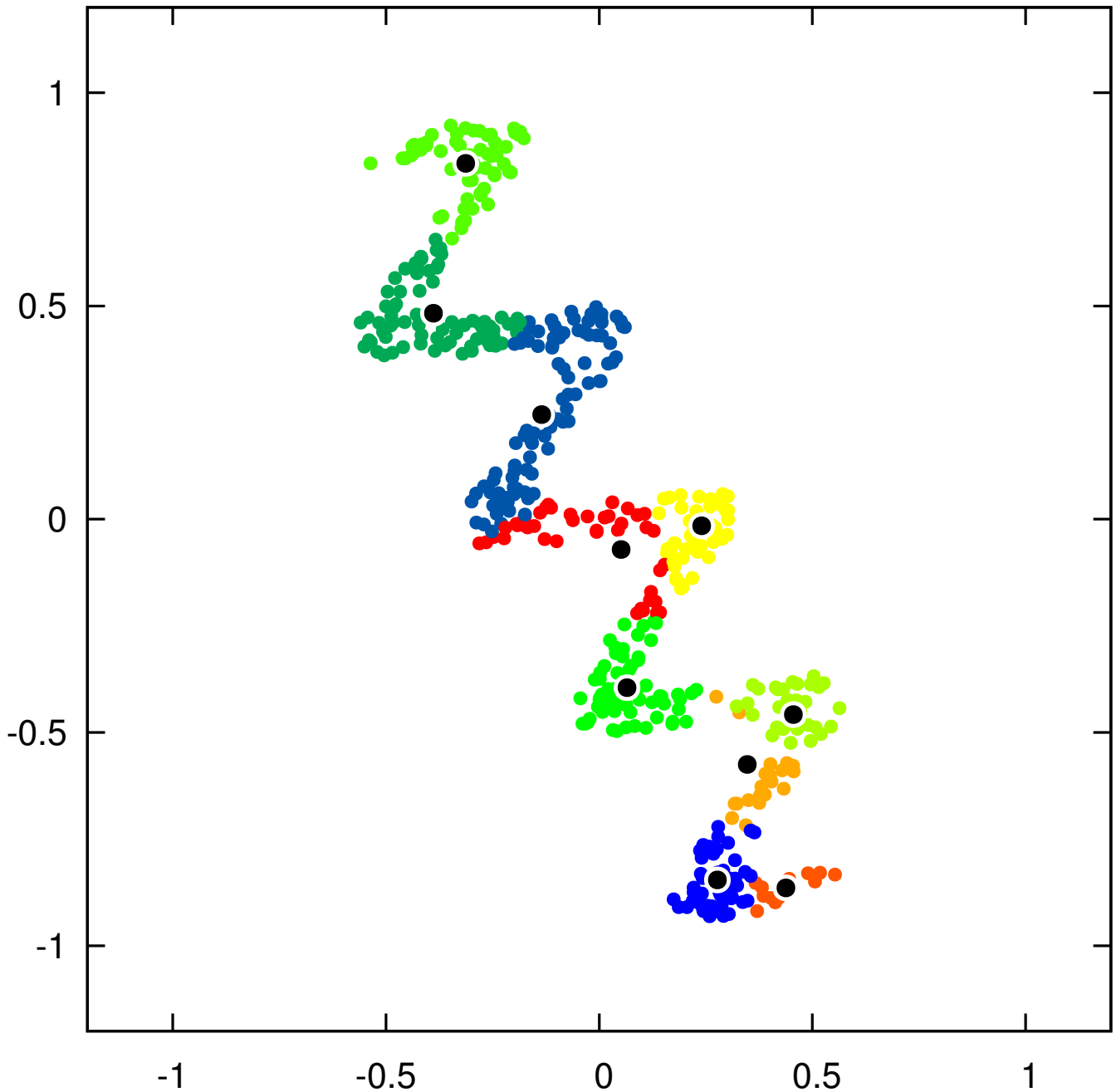

1
0.5
0
-0.5
-1
-1
-0.5
0
0.5
1

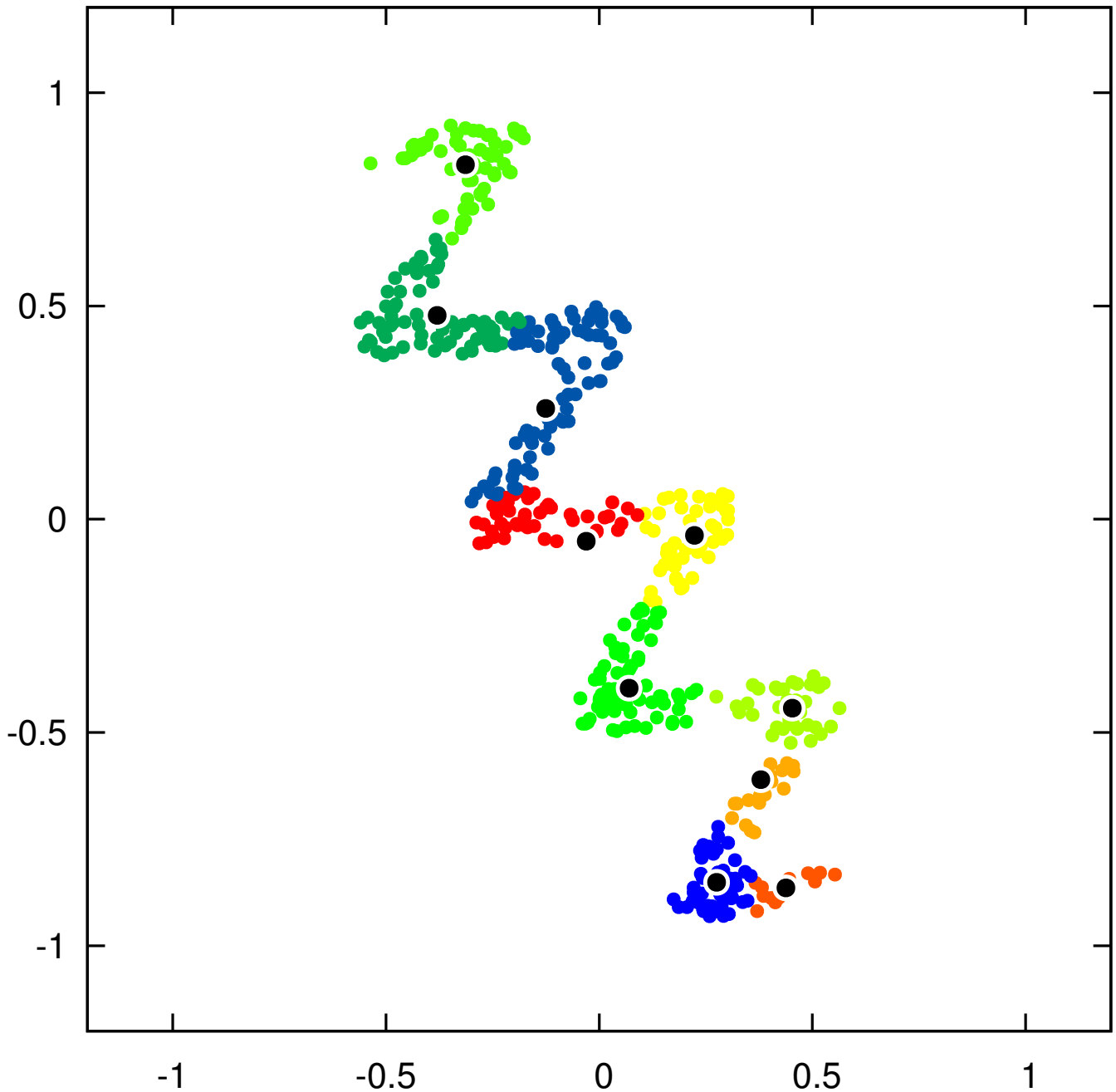

1
0.5
0
-0.5
-1
-1
-0.5
0
0.5
1

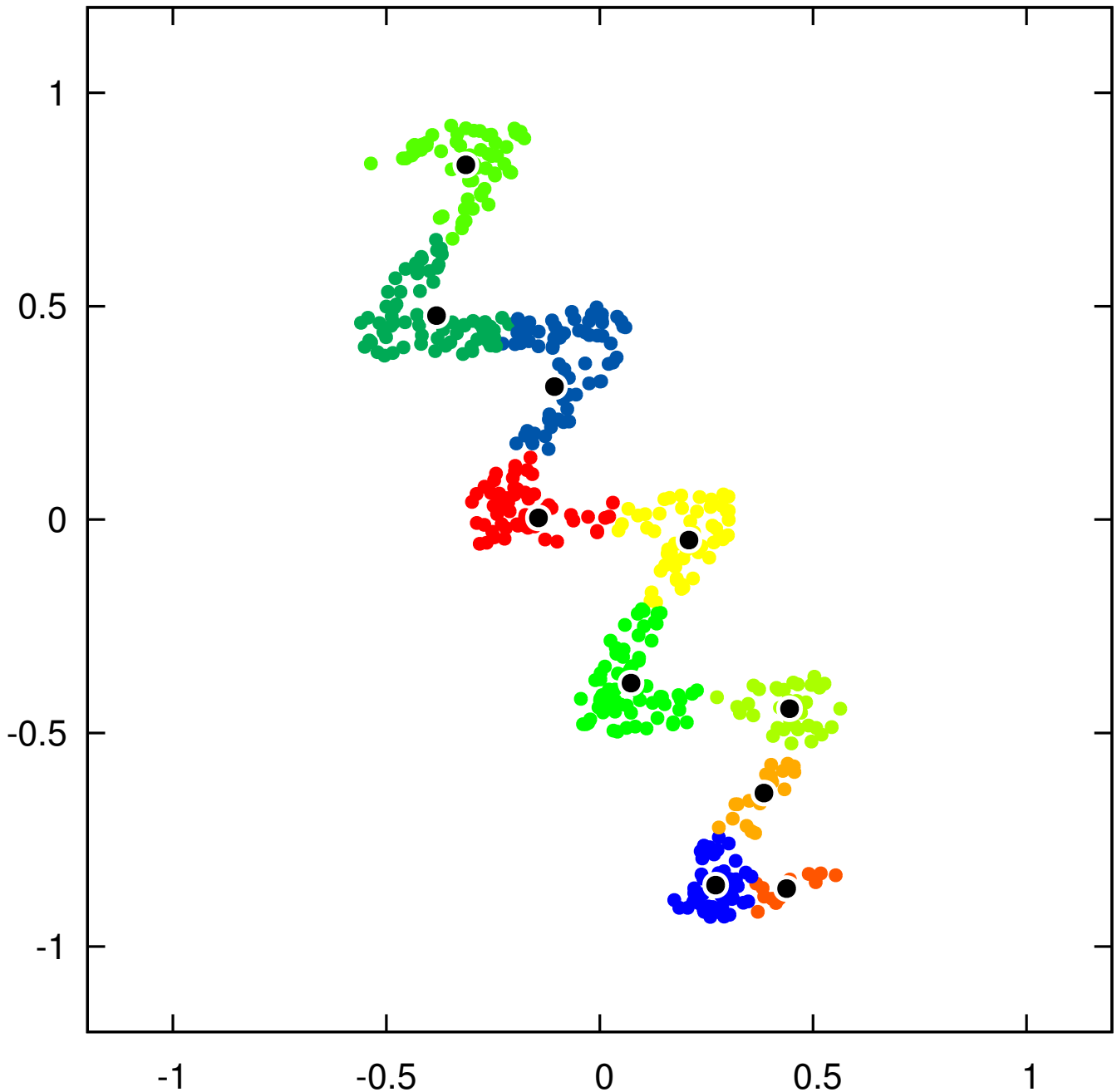

1
0.5
0
-0.5
-1
-1
-0.5
0
0.5
1

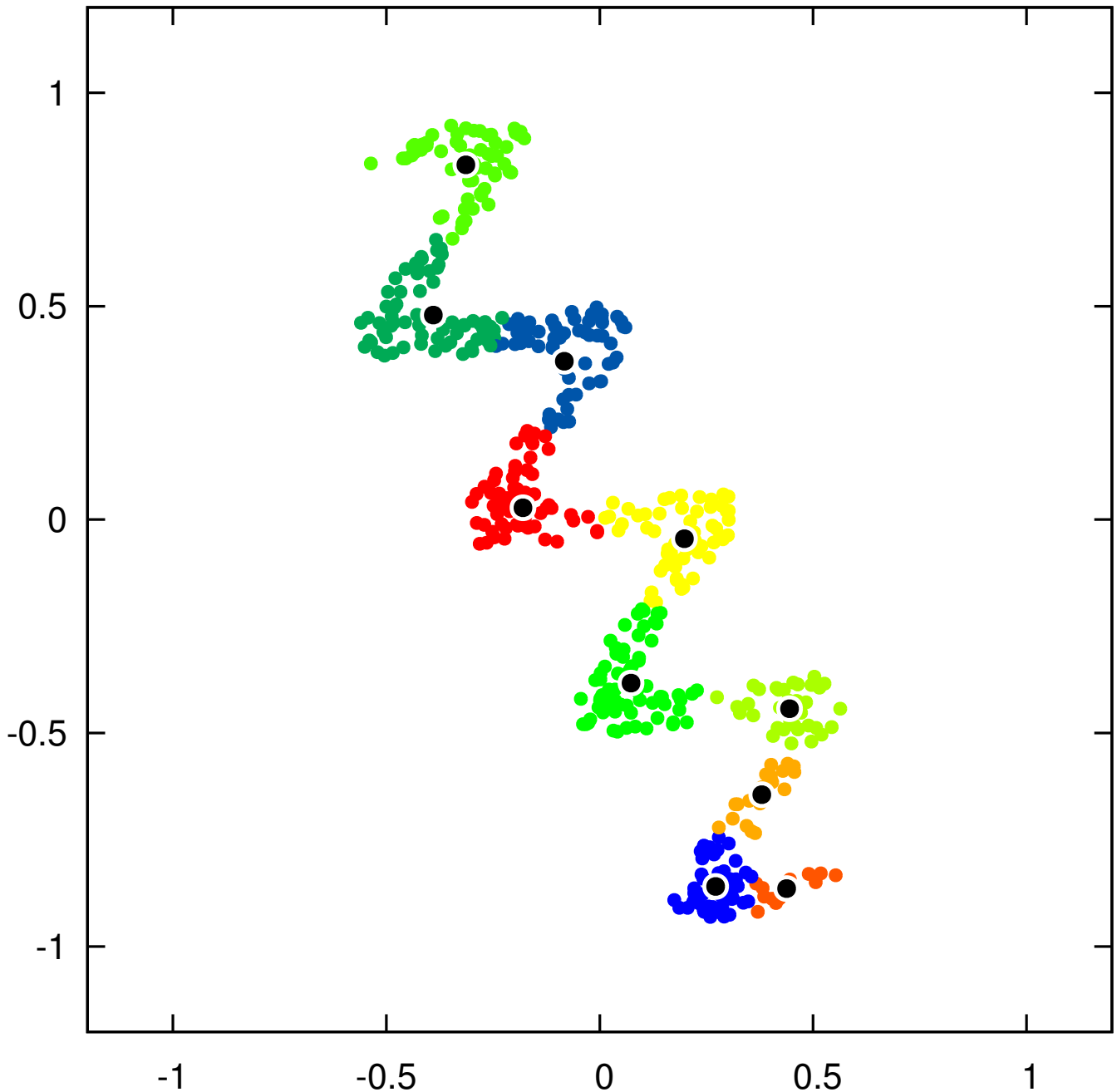

1
0.5
0
-0.5
-1
-1
-0.5
0
0.5
1

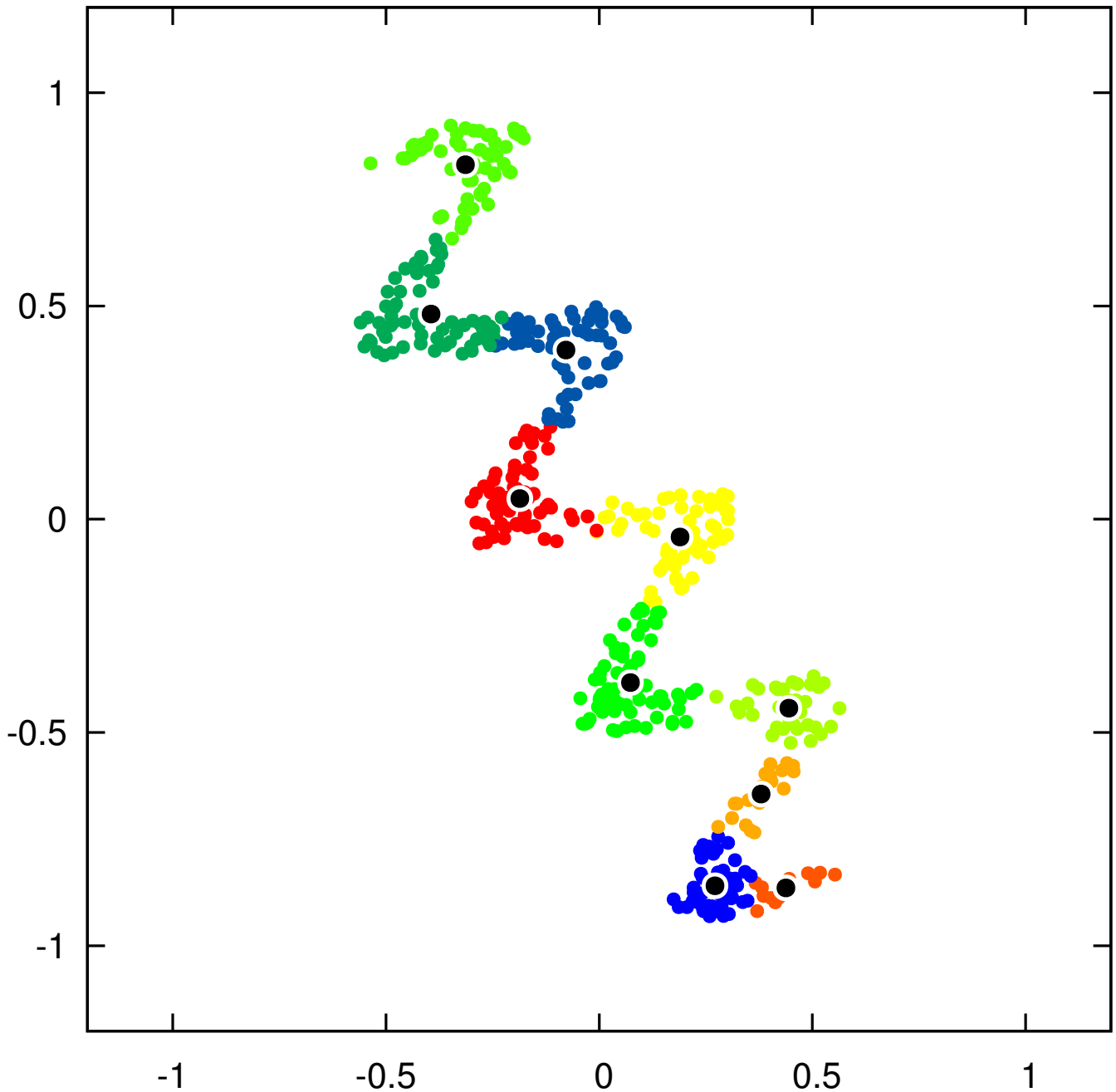

1
0.5
0
-0.5
-1
-1
-0.5
0
0.5
1

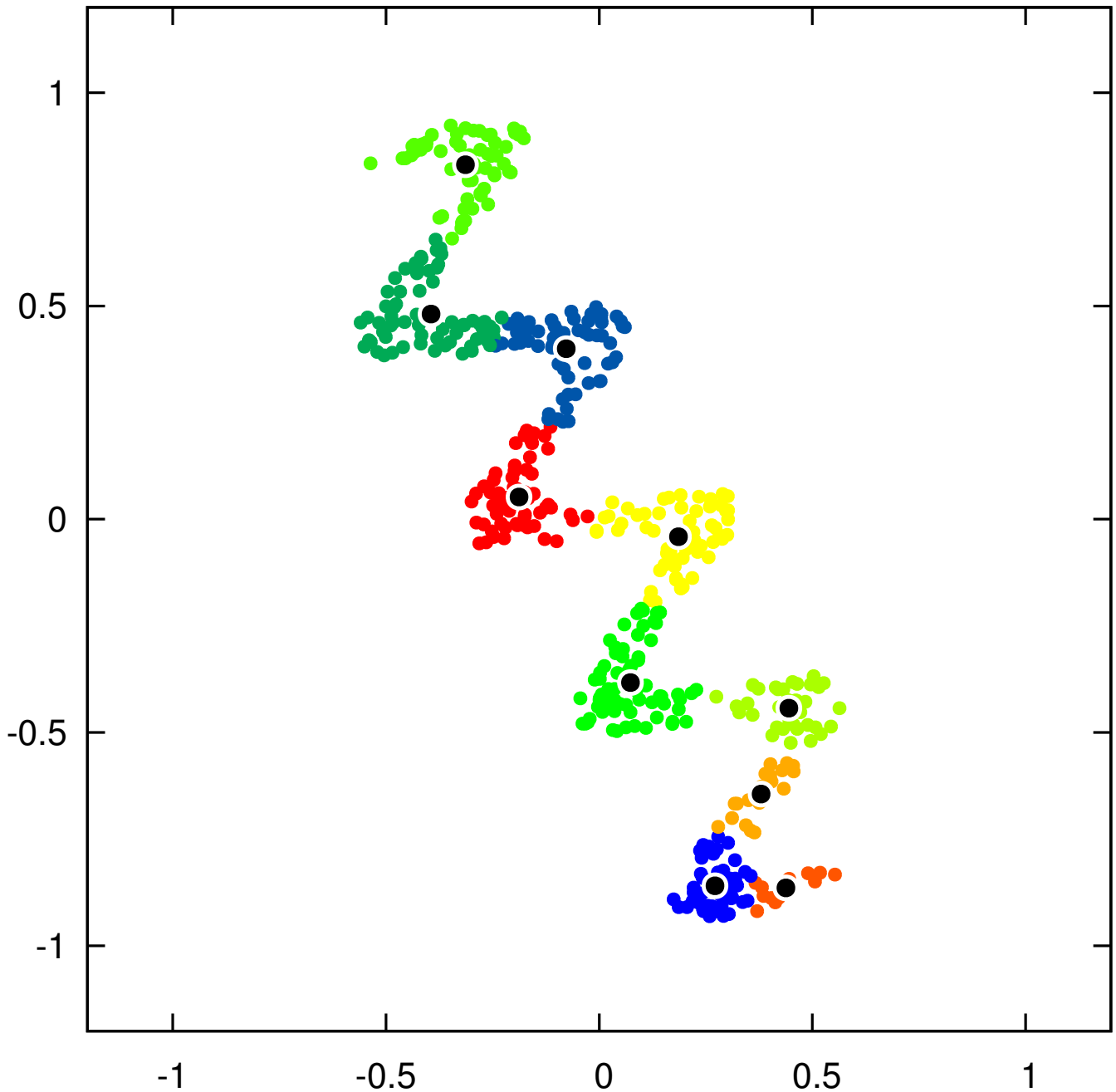

1
0.5
0
-0.5
-1
-1
-0.5
0
0.5
1

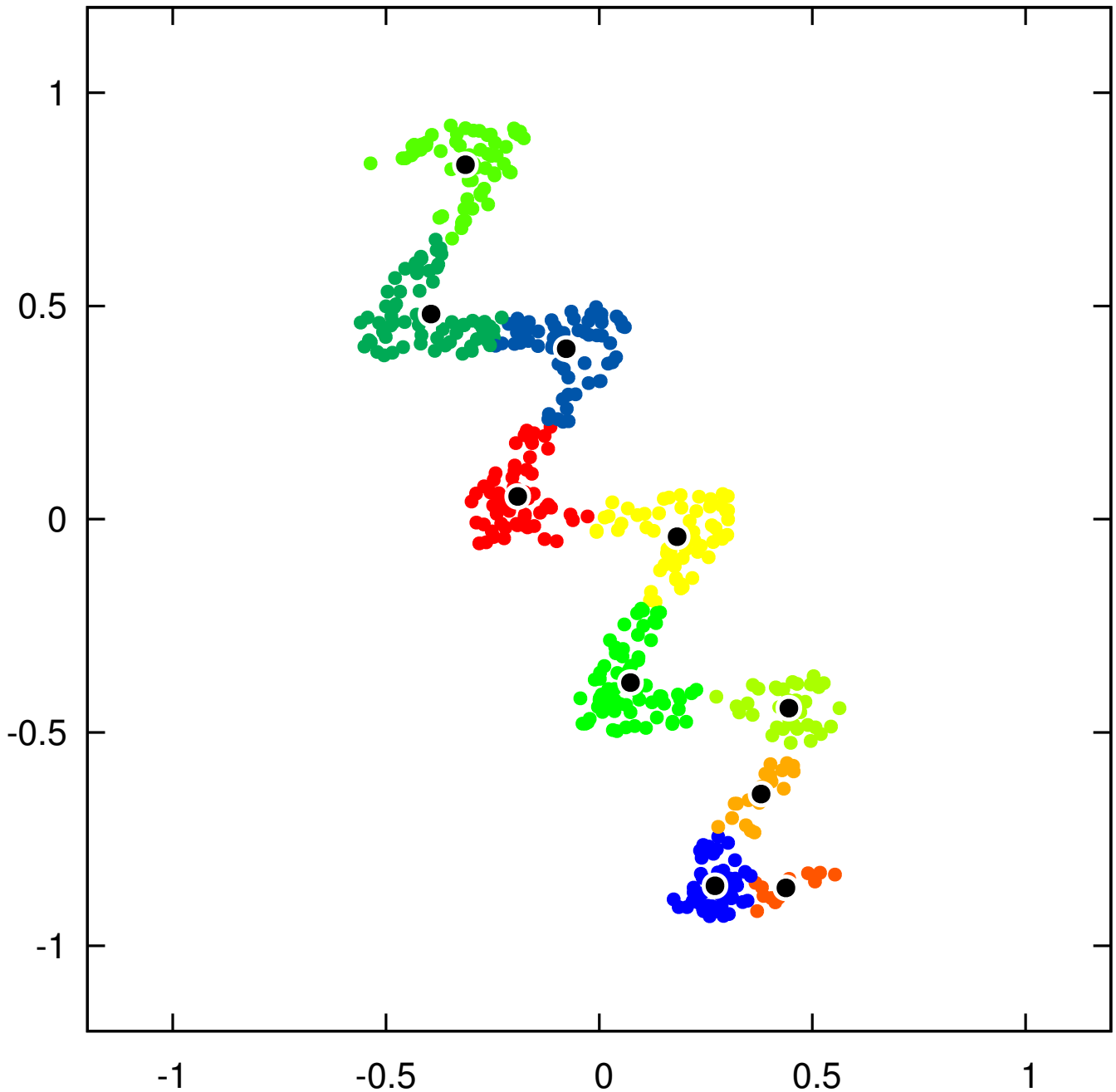

1
0.5
0
-0.5
-1
-1
-0.5
0
0.5
1


1
0.5
0
-0.5
-1
-1
-0.5
0
0.5
1


1
0.5
0
-0.5
-1
-1
-0.5
0
0.5
1


1
0.5
0
-0.5
-1
-1
-0.5
0
0.5
1


1
0.5
0
-0.5
-1
-1
-0.5
0
0.5
1


1
0.5
0
-0.5
-1
-1
-0.5
0
0.5
1

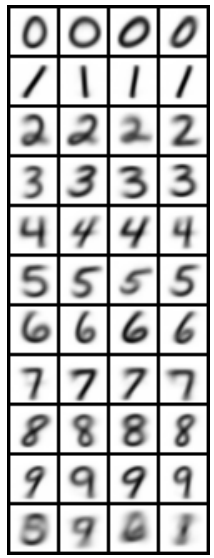
We can apply that algorithm to images from MNIST ($28 \times 28 \times 1$) or CIFAR ($32 \times 32 \times 3$) by considering them as vectors from $\mathbb{R}^{784}$ and $\mathbb{R}^{3072}$ respectively.

Centroids can similarly be visualized as images.

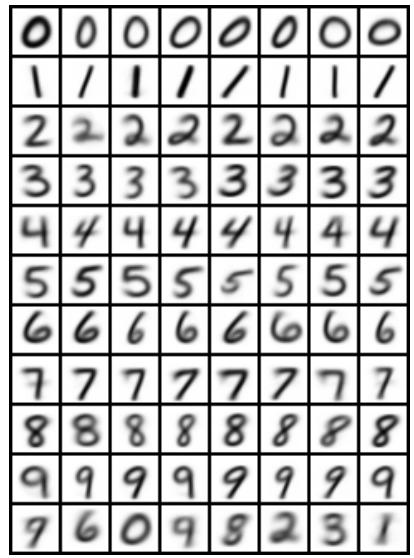
Clustering can be done per-class, or for all the class mixed.

$K = 1$

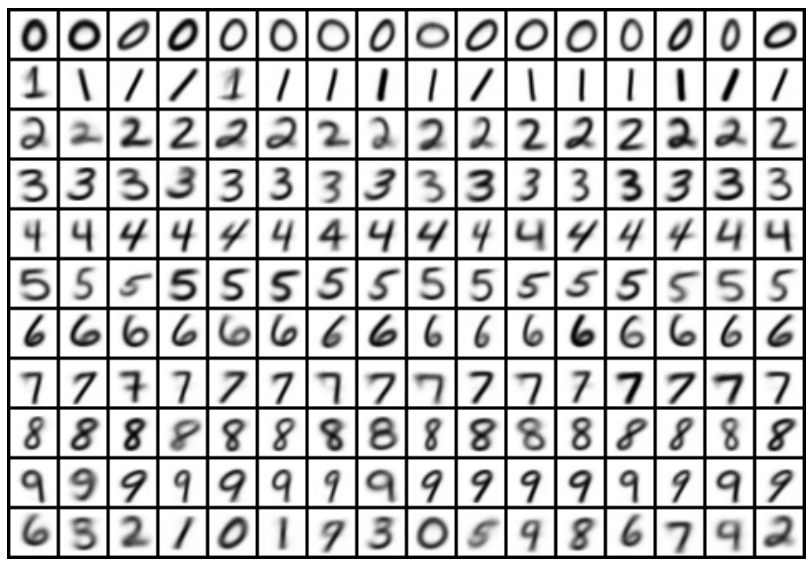
$K = 2$



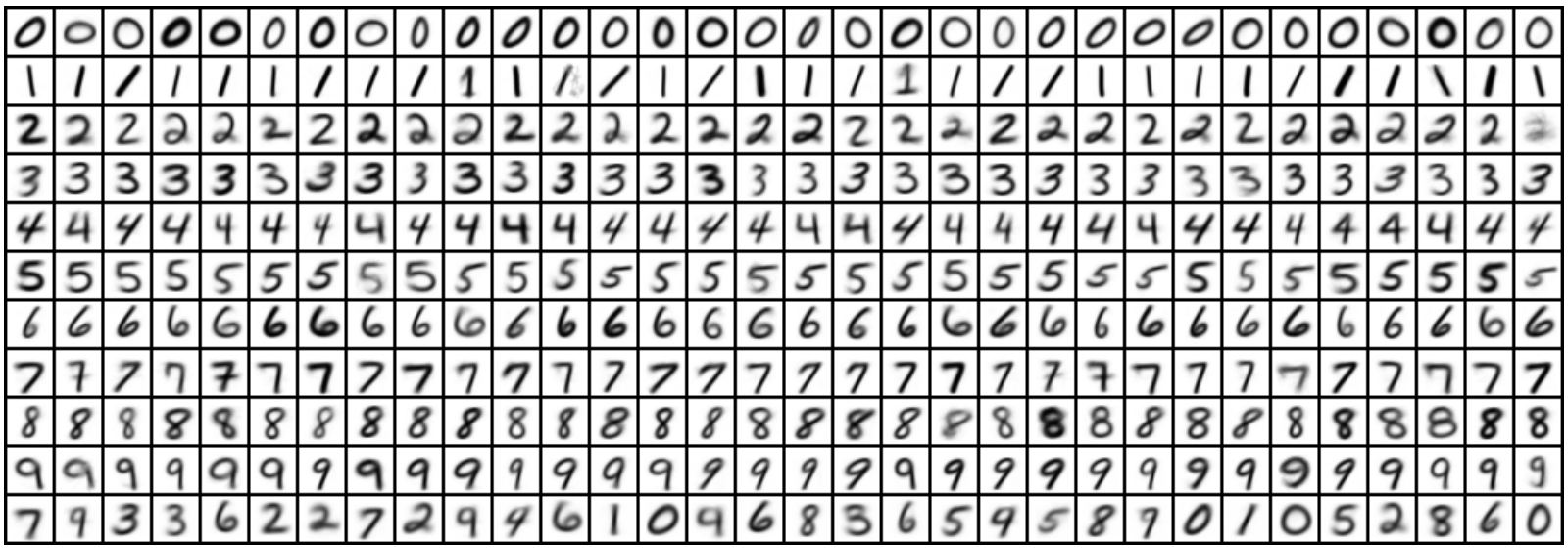
$K = 4$



$K = 8$



$K = 16$



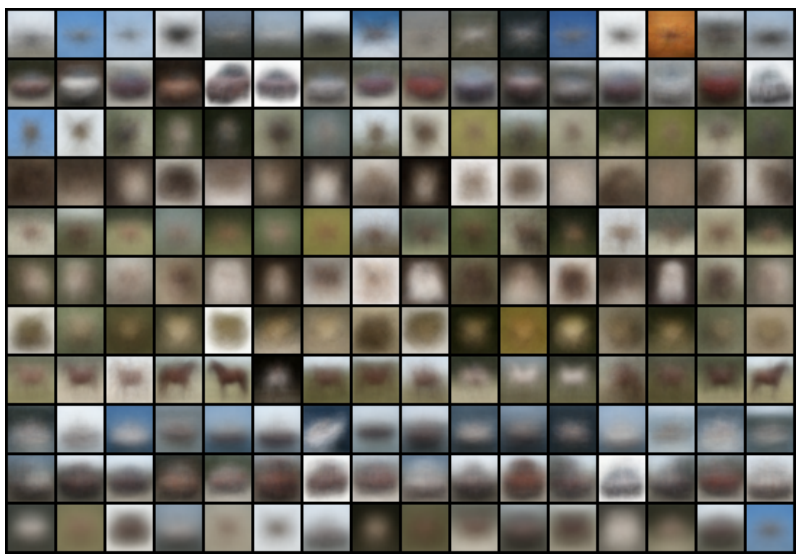
$K = 32$

$K = 1$

$K = 2$

$K = 4$

$K = 8$



$K = 16$

$K = 32$

The Principal Component Analysis (PCA) aims also at extracting an information in a L^2 sense. Instead of clusters, it looks for an "affine subspace", *i.e.* a point and a basis, that spans the data.

Given data-points

$$x_n \in \mathbb{R}^D, \ n = 1, \ldots, N$$

(A) compute the average and center the data

$$\begin{aligned} \bar{x} &= \frac{1}{N} \sum_n x_n \\ \forall n, \ x_n^{(0)} &= x_n - \bar{x} \end{aligned}$$

The Principal Component Analysis (PCA) aims also at extracting an information in a L^2 sense. Instead of clusters, it looks for an "affine subspace", *i.e.* a point and a basis, that spans the data.

Given data-points

$$x_n \in \mathbb{R}^D, \ n = 1, \ldots, N$$

(A) compute the average and center the data

$$\begin{aligned} \bar{x} &= \frac{1}{N} \sum_n x_n \\ \forall n, \ x_n^{(0)} &= x_n - \bar{x} \end{aligned}$$

and then for $t = 1, \ldots, D$,

(B) pick the direction and project the data

$$\begin{aligned} v_t &= \underset{\|v\|=1}{\operatorname{argmax}} \sum_n \left(v \cdot x_n^{(t-1)} \right)^2 \\ \forall n, \ x_n^{(t)} &= x_n^{(t-1)} - \left(v_t \cdot x_n^{(t-1)} \right) v_t. \end{aligned}$$

Although this is a simple way to envision PCA, standard implementations rely on an eigendecomposition. With

$$X = \begin{pmatrix} — & x_1 & — \\ & \vdots & \\ — & x_N & — \end{pmatrix}$$

we have

$$\sum_n (v \cdot x_n)^2$$

Although this is a simple way to envision PCA, standard implementations rely on an eigendecomposition. With

$$X = \begin{pmatrix} — & x_1 & — \\ & \vdots & \\ — & x_N & — \end{pmatrix}$$

we have

$$\sum_n (v \cdot x_n)^2 = \left\| \begin{pmatrix} v \cdot x_1 \\ \vdots \\ v \cdot x_N \end{pmatrix} \right\|_2^2$$

Although this is a simple way to envision PCA, standard implementations rely on an eigendecomposition. With

$$X = \begin{pmatrix} — & x_1 & — \\ & \vdots & \\ — & x_N & — \end{pmatrix}$$

we have

$$\begin{aligned} \sum_n (v \cdot x_n)^2 &= \left\| \begin{pmatrix} v \cdot x_1 \\ \vdots \\ v \cdot x_N \end{pmatrix} \right\|_2^2 \\ &= \left\| v X^T \right\|_2^2 \end{aligned}$$

Although this is a simple way to envision PCA, standard implementations rely on an eigendecomposition. With

$$X = \begin{pmatrix} — & x_1 & — \\ & \vdots & \\ — & x_N & — \end{pmatrix}$$

we have

$$\begin{aligned} \sum_n (v \cdot x_n)^2 &= \left\| \begin{pmatrix} v \cdot x_1 \\ \vdots \\ v \cdot x_N \end{pmatrix} \right\|_2^2 \\ &= \left\| v X^T \right\|_2^2 \\ &= (v X^T)(v X^T)^T \end{aligned}$$

Although this is a simple way to envision PCA, standard implementations rely on an eigendecomposition. With

$$X = \begin{pmatrix} — & x_1 & — \\ & \vdots & \\ — & x_N & — \end{pmatrix}$$

we have

$$\begin{aligned} \sum_n (v \cdot x_n)^2 &= \left\| \begin{pmatrix} v \cdot x_1 \\ \vdots \\ v \cdot x_N \end{pmatrix} \right\|_2^2 \\ &= \left\| v X^T \right\|_2^2 \\ &= (v X^T)(v X^T)^T \\ &= v(X^T X) v^T. \end{aligned}$$

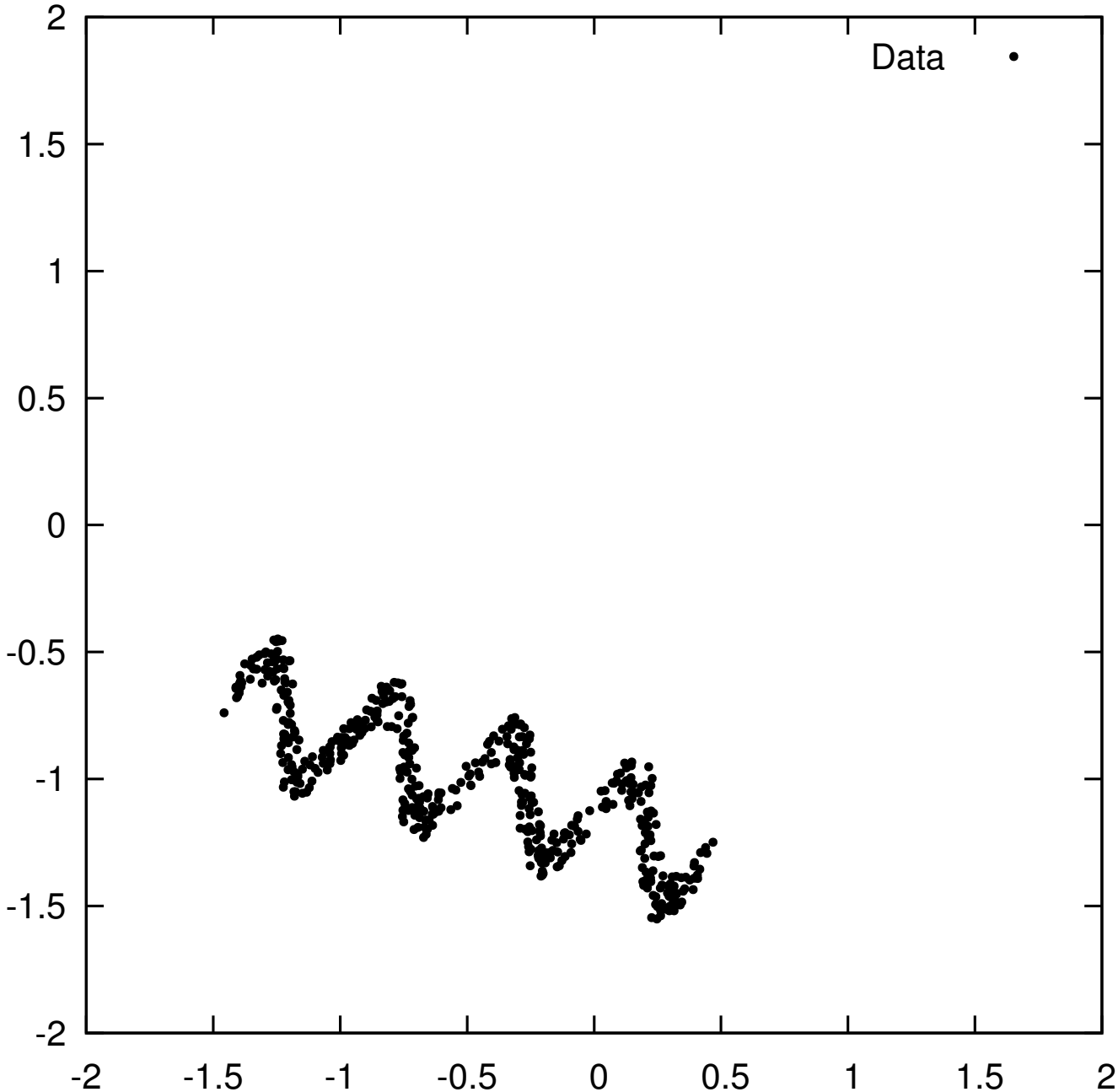
Although this is a simple way to envision PCA, standard implementations rely on an eigendecomposition. With

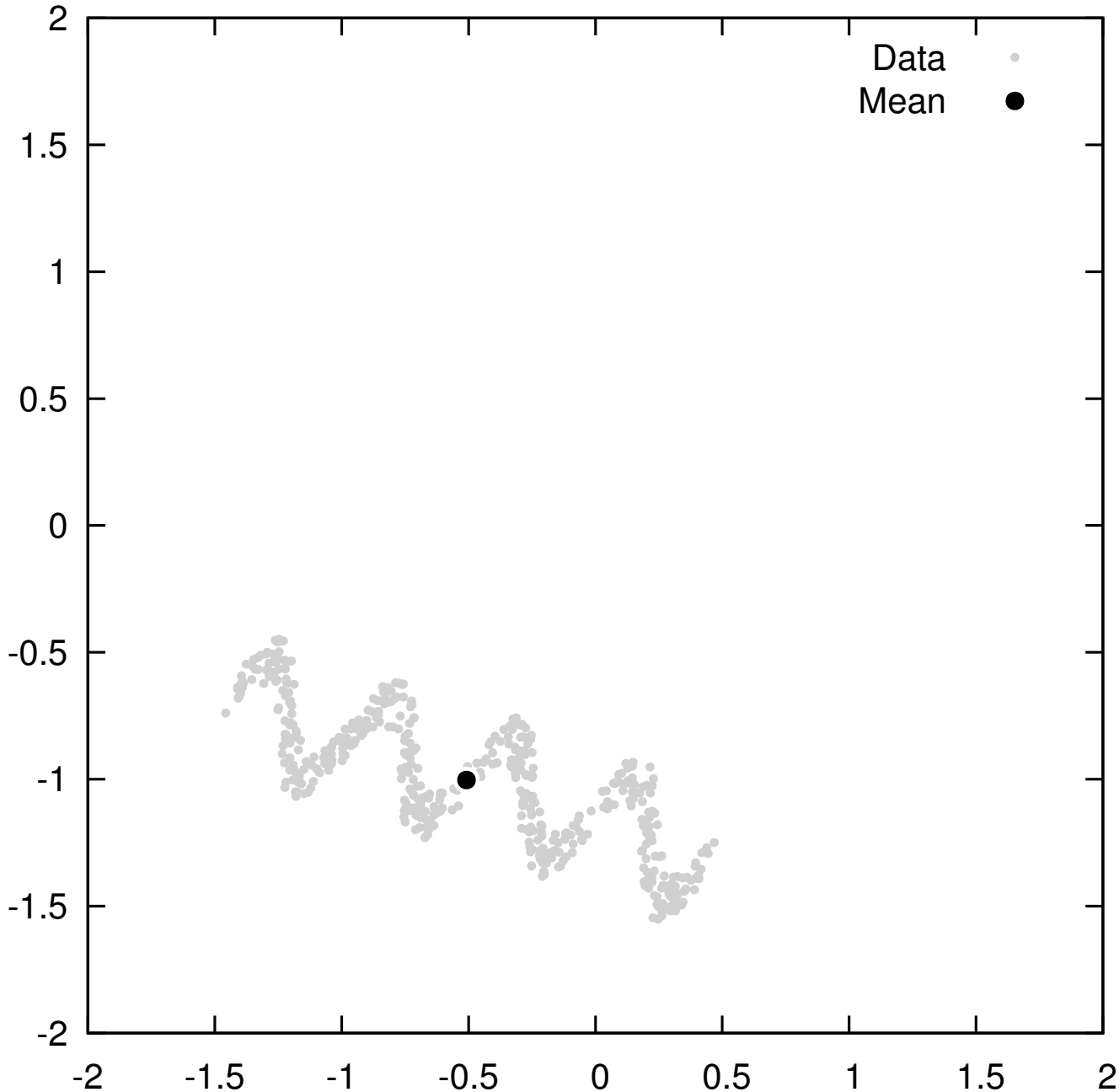
$$X = \begin{pmatrix} — & x_1 & — \\ & \vdots & \\ — & x_N & — \end{pmatrix}$$

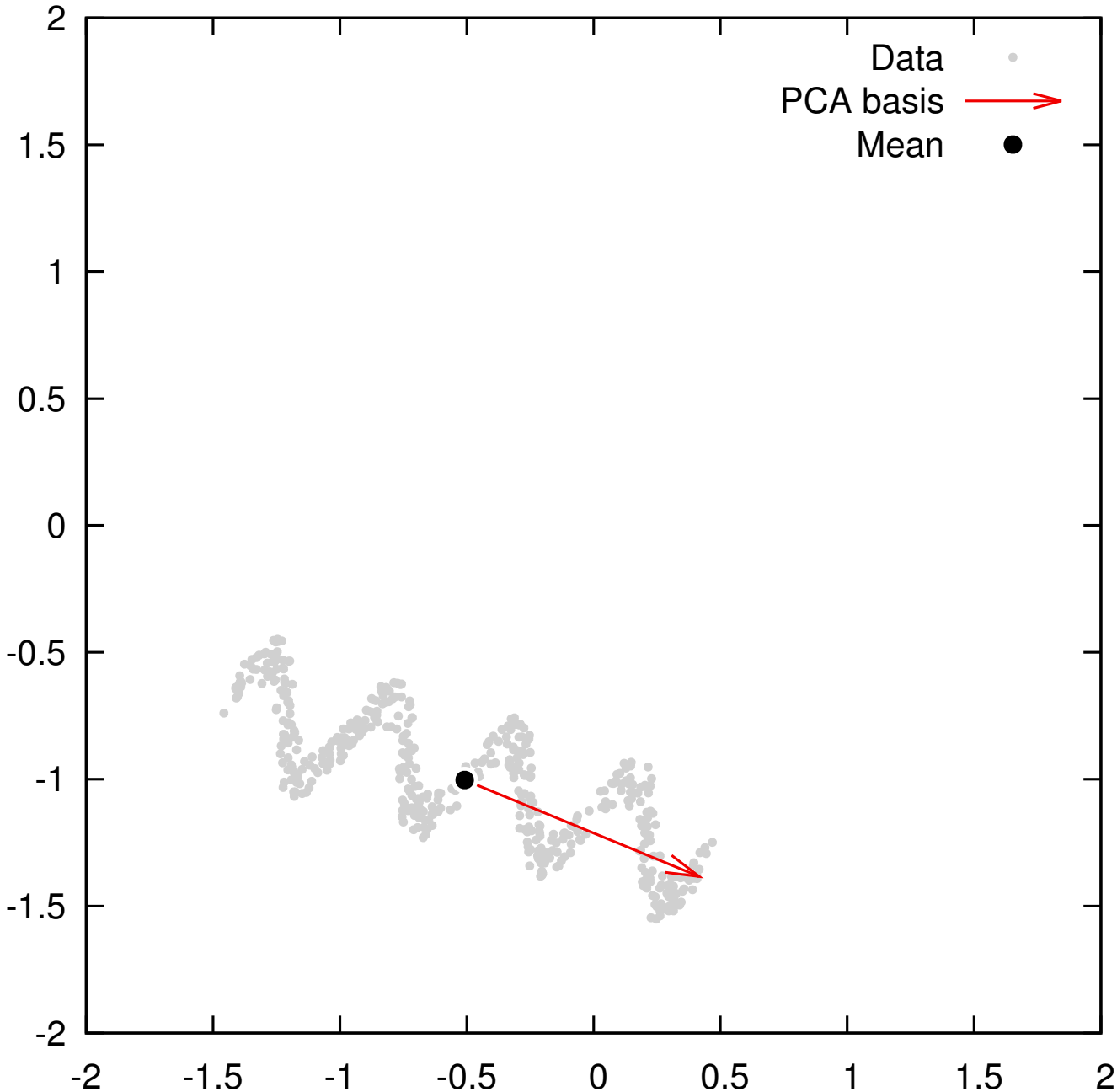
we have

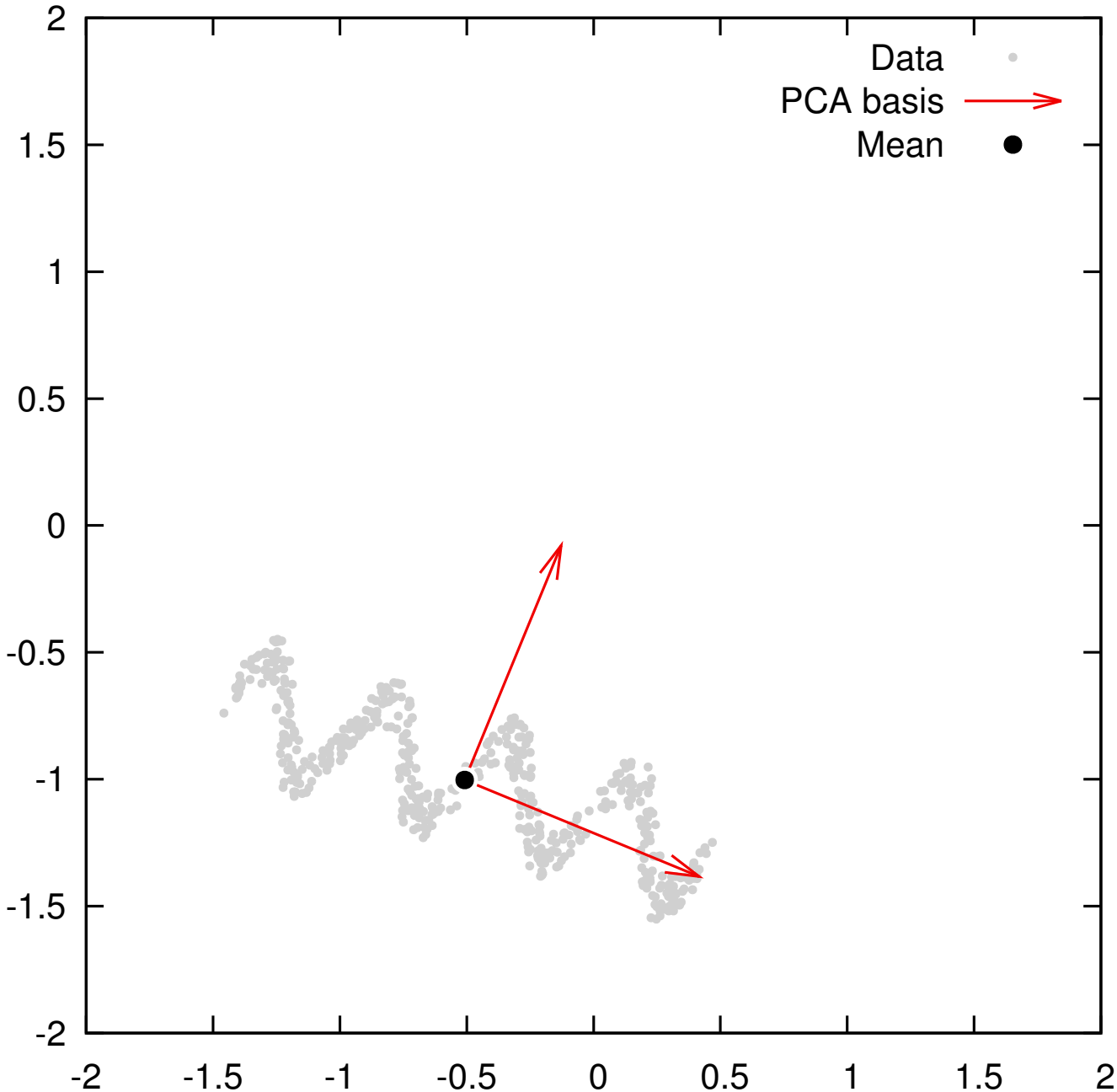
$$\begin{aligned} \sum_n (v \cdot x_n)^2 &= \left\| \begin{pmatrix} v \cdot x_1 \\ \vdots \\ v \cdot x_N \end{pmatrix} \right\|_2^2 \\ &= \left\| v X^T \right\|_2^2 \\ &= (v X^T)(v X^T)^T \\ &= v(X^T X) v^T. \end{aligned}$$

From this we can derive that $v_1, v_2, \ldots, v_D$ are the eigenvectors of $X^T X$ ranked according to [the absolute values of] their eigenvalues.


Data
2
1.5
1
0.5
0
-0.5
-1
-1.5
-2
-2
-1.5
-1
-0.5
0
0.5
1
1.5
2


Data
Mean
2
1.5
1
0.5
0
-0.5
-1
-1.5
-2
-2
-1.5
-1
-0.5
0
0.5
1
1.5
2

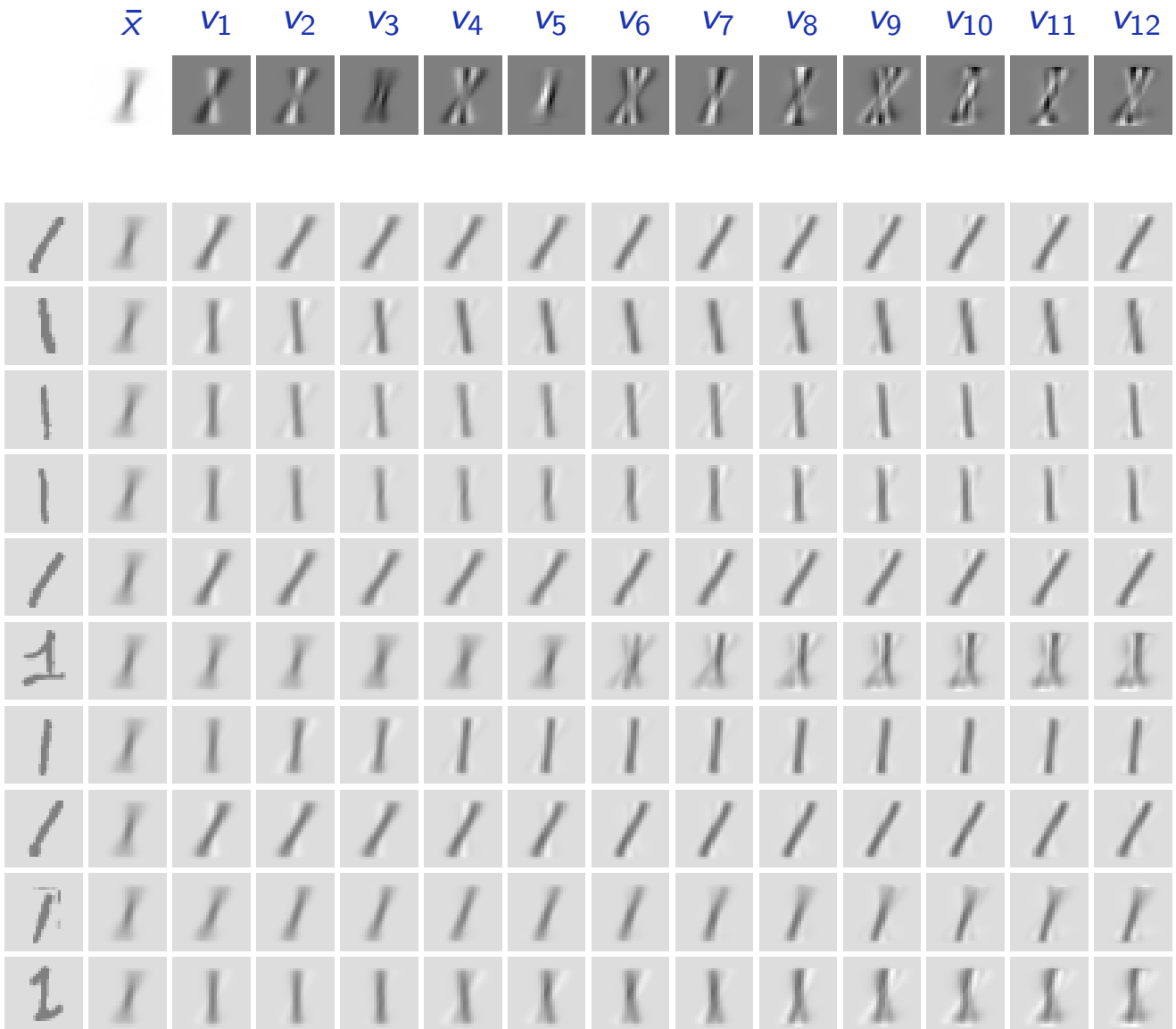

Data
PCA basis
Mean
2
1.5
1
0.5
0
-0.5
-1
-1.5
-2
-2
-1.5
-1
-0.5
0
0.5
1
1.5
2

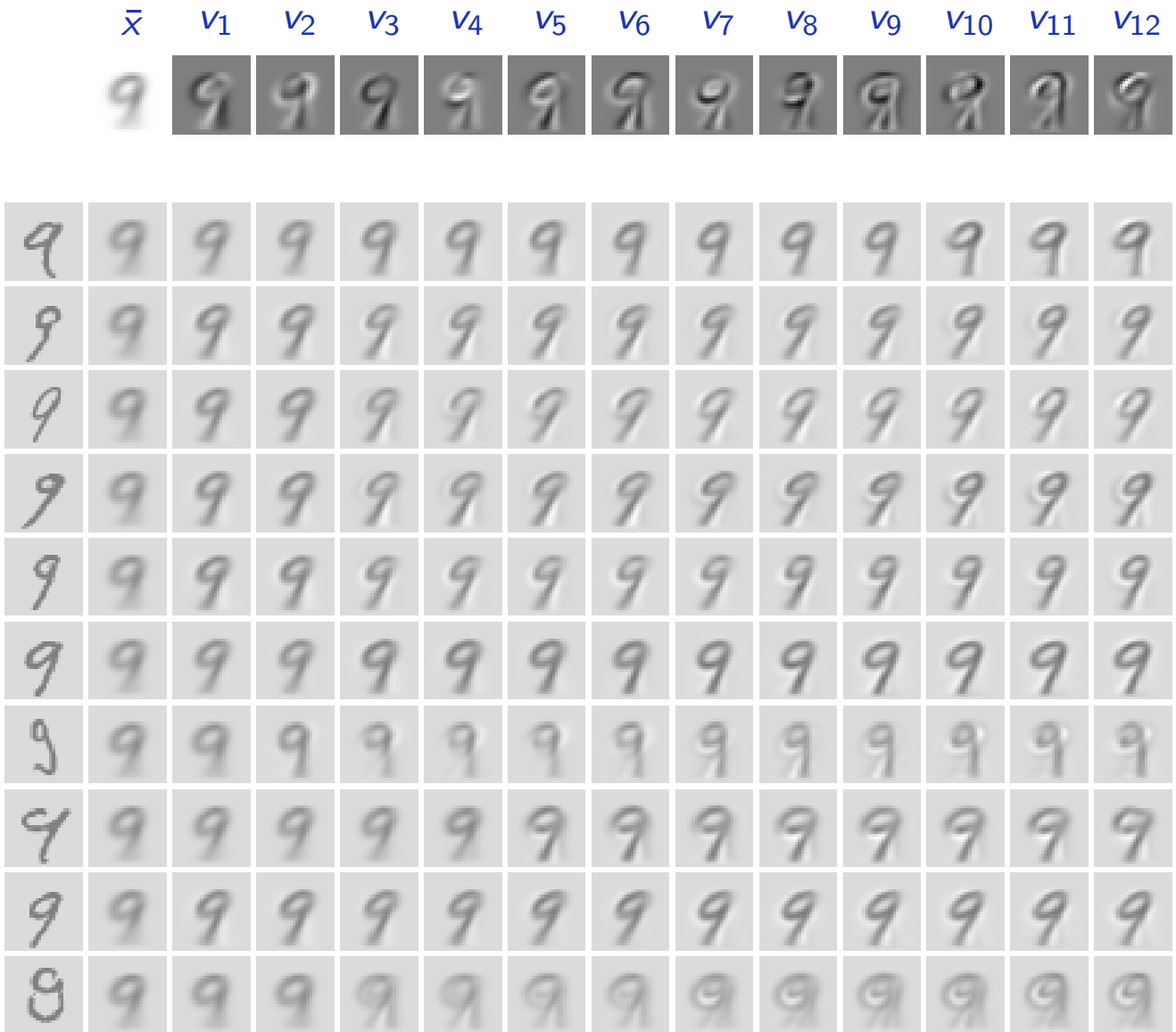

Data
PCA basis
Mean
2
1.5
1
0.5
0
-0.5
-1
-1.5
-2
-2
-1.5
-1
-0.5
0
0.5
1
1.5
2

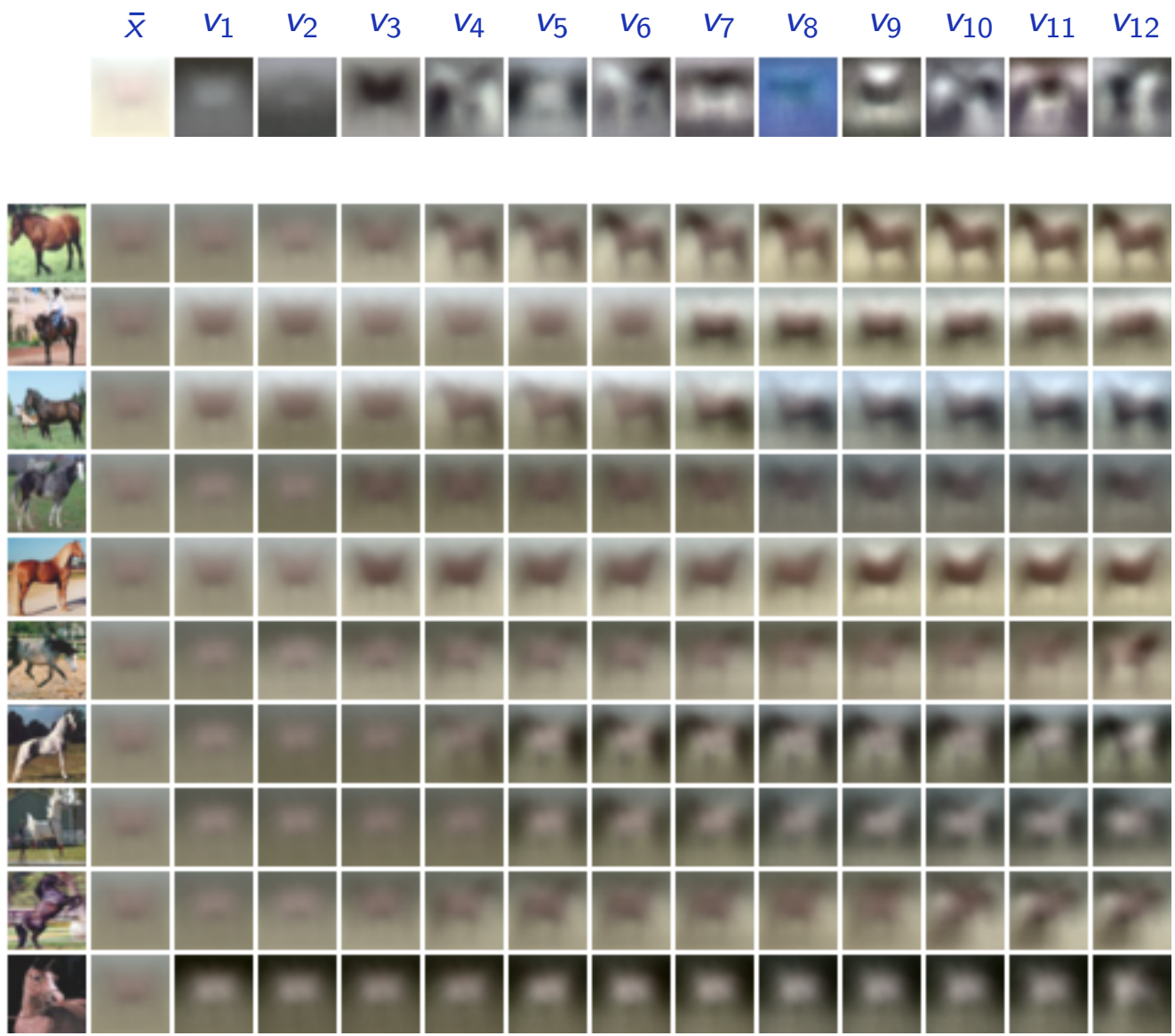
As for K-means, we can apply that algorithm to images from MNIST or CIFAR by considering them as vectors.

For any sample x and any T, we can compute a reconstruction using T vectors from the PCA basis, *i.e.*

$$\bar{x} + \sum_{t=1}^{T} (v_t \cdot x) v_t.$$


$\bar{x}$
v_1
v_2
v_3
v_4
v_5
v_6
v_7
v_8
v_9
v_{10}
v_{11}
v_{12}


$\bar{x}$
v_1
v_2
v_3
v_4
v_5
v_6
v_7
v_8
v_9
v_{10}
v_{11}
v_{12}


$\bar{x}$
v_1
v_2
v_3
v_4
v_5
v_6
v_7
v_8
v_9
v_{10}
v_{11}
v_{12}


$\bar{x}$
v_1
v_2
v_3
v_4
v_5
v_6
v_7
v_8
v_9
v_{10}
v_{11}
v_{12}

These results show that even crude embeddings capture something meaningful. Changes in pixel intensity as expected, but also deformations in the "indexing" space (*i.e.* the image plan).

However, translations and deformations damage the representation badly. "Composition" (*e.g.* object on background) is not handled at all.

These strengths and shortcomings provide an intuitive motivation for "deep neural networks", and the rest of this course.

We would like

- to use many encoding "of these sorts" for small local structures with limited variability,
- have different "channels" for different components,
- process at multiple scales.

Computationally, we would like to deal with large signals and large training sets, so we need to avoid super-linear cost in one or the other.

Practical session:

https://fleuret.org/dlc/dlc-practical-2.pdf

The end

References

Y. Bengio and Y. Grandvalet. No unbiased estimator of the variance of k-fold cross-validation. *Journal of Machine Learning Research (JMLR)*, 5:1089–1105, 2004.

S. Geman and E. Bienenstock. Neural networks and the bias/variance dilemma. *Neural Computation*, 4:1–58, 1992.

A. Krizhevsky. Learning multiple layers of features from tiny images. Master's thesis, Department of Computer Science, University of Toronto, 2009.

V. N. Vapnik. *The Nature of Statistical Learning Theory*. Springer-Verlag, New York, 1995.