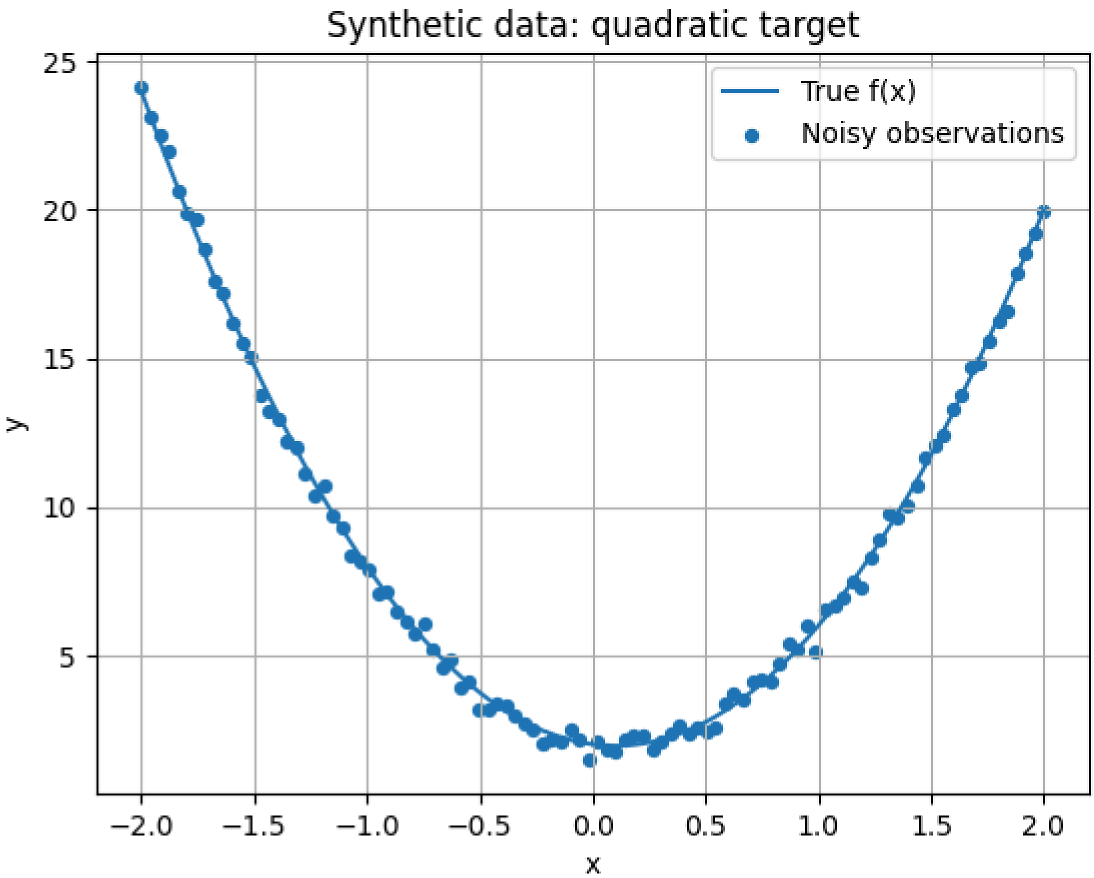
# Week 37 Excercises

Initializing project with imports and a global variable for reproducibility

In [3]:
```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import mean_squared_error

SEED = 42
np.random.seed(SEED)
```

Initializing and visualizing target function

In [4]:
```
# Data generation
n = 100
x = np.linspace(-2, 2, n)
def f(x): return 2 - x + 5*x**2

noise_sigma = 0.3
y_true = f(x)
y = y_true + np.random.normal(0, noise_sigma, size=n)

plt.figure()
plt.plot(x, y_true, label="True f(x)")
plt.scatter(x, y, s=18, label="Noisy observations")
plt.title("Synthetic data: quadratic target")
plt.xlabel("x");
plt.ylabel("y");
plt.legend();
plt.grid()
plt.show()
```




# Exercise 1

We build a degree-2 (extendable) polynomial design matrix and **standardize** each feature to mean 0 and variance 1.

We also **center** the target to mean 0 so we can omit an explicit intercept term.

In [ ]:
```
def polynomial_features(x: np.ndarray, degree: int, intercept: bool=False) -> np
    x = np.asarray(x).reshape(-1)
    X = np.column_stack([x**k for k in range(1 if not intercept else 0, degree+1
    if intercept:
        X[:,0] = 1.0
    return X

degree = 2
X = polynomial_features(x, degree=degree, intercept=False)

X_mean = X.mean(axis=0)
X_std = X.std(axis=0)
X_std[X_std == 0] = 1.0
X_s = (X - X_mean) / X_std
#Centering
y_mean = y.mean()
y_c = y - y_mean
```

**Do we need to center the *x* values?**

Not strictly, what matters for fair regularization is scaling the columns of the design matrix, centering *x* itself is also indirectly handled by standardizing the constructed features.

# Exercise 2

Let $J_{\text{OLS}}(\theta) = \frac{1}{n}\|y - X\theta\|_2^2$. Then

$$\nabla_\theta J_{\text{OLS}} = -\frac{2}{n}X^\top(y - X\theta) = \frac{2}{n}(X^\top X\theta - X^\top y).$$

For Ridge with

$$J_{\text{Ridge}}(\theta) = \frac{1}{n}\|y - X\theta\|_2^2 + \lambda\,\|\theta\|_2^2$$

for linearity we just add the derivative of the additional term $2\lambda\theta$:

$$\nabla_\theta J_{\text{Ridge}} = \frac{2}{n}(X^\top X\theta - X^\top y) + 2\lambda\theta.$$

# Exercise 3

**a)** We use the standardized (X) and centered (y). Closed forms:

$$\hat{\theta}_{\text{OLS}} = (X^\top X)^{-1}X^\top y, \quad \hat{\theta}_{\text{Ridge}} = (X^\top X + \lambda I)^{-1}X^\top y.$$

In [6]:

```
from numpy.linalg import inv

def closed_form_ols(Xs: np.ndarray, y_c: np.ndarray) -> np.ndarray:
    XT_X = Xs.T @ Xs
    return inv(XT_X) @ (Xs.T @ y_c)

def closed_form_ridge(Xs: np.ndarray, y_c: np.ndarray, lam: float) -> np.ndarray
    XT_X = Xs.T @ Xs
    p = XT_X.shape[0]
    return inv(XT_X + lam * np.eye(p)) @ (Xs.T @ y_c)

lam_values = [0.0, 1e-4, 1e-2, 1e-1]
theta_ols_cf = closed_form_ols(X_s, y_c)
thetas_ridge_cf = {lam: closed_form_ridge(X_s, y_c, lam) for lam in lam_values i

print("Closed-form OLS theta:", theta_ols_cf)
for lam, th in thetas_ridge_cf.items():
    print(f"Closed-form Ridge theta (λ={lam}):", th)
```

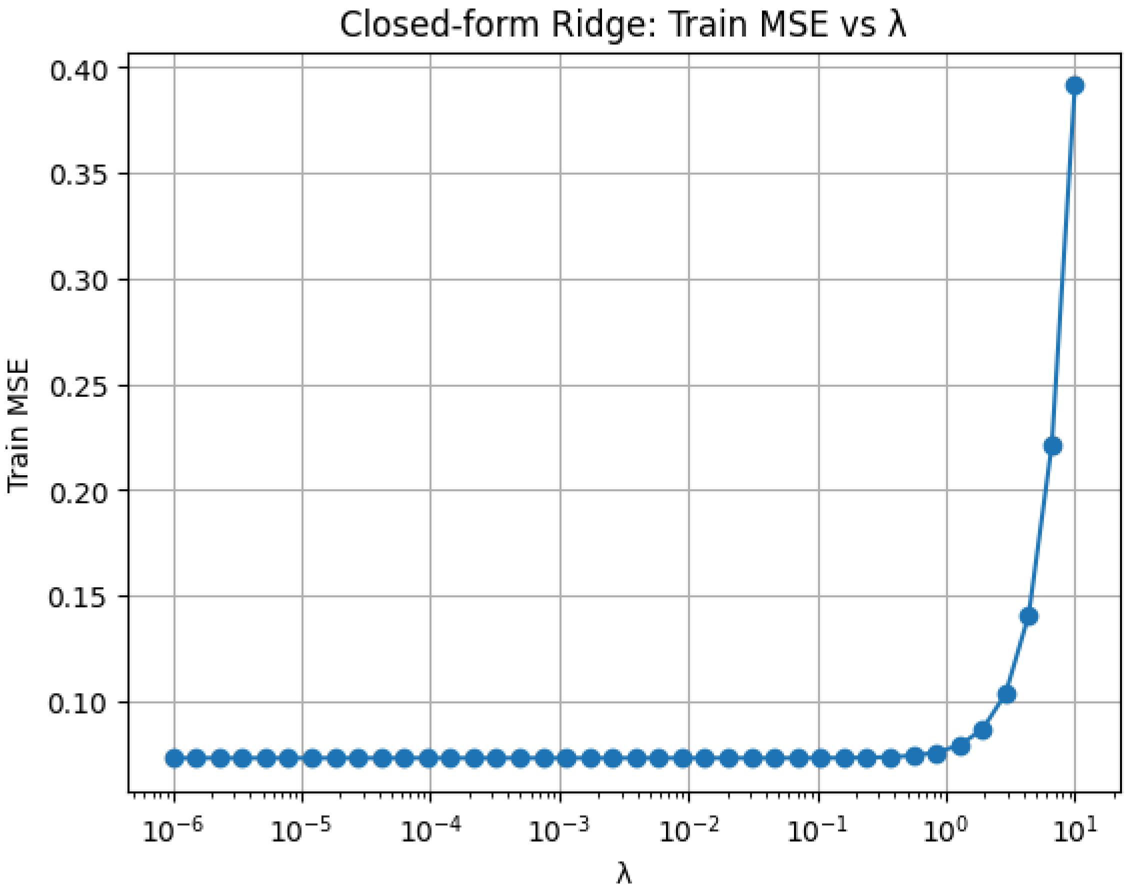
```
Closed-form OLS theta: [-1.15424048  6.09911638]
Closed-form Ridge theta (λ=0.0001): [-1.15423933  6.09911028]
Closed-form Ridge theta (λ=0.01): [-1.15412507  6.09850653]
Closed-form Ridge theta (λ=0.1): [-1.1530874   6.09302335]
```

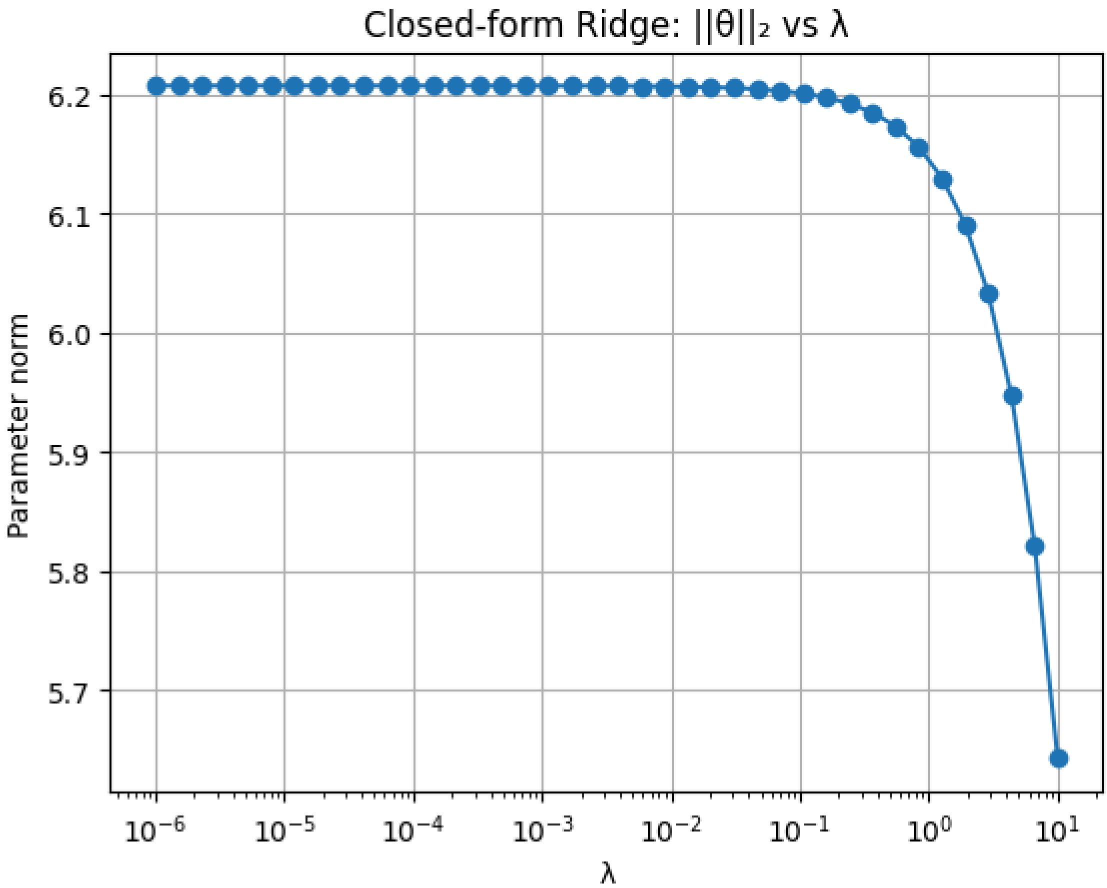
**3b)** Explore as a function of λ We compare training MSE and parameter norms across λ.

In [7]:

```
def predict_from_theta(Xs, theta, y_offset=0.0):
    return Xs @ theta + y_offset

def score_theta(Xs, y, theta, y_offset=0.0):
    yhat = predict_from_theta(Xs, theta, y_offset)
    return mean_squared_error(y, yhat)
```

```
lambdas = np.logspace(-6, 1, 40)
mse_train_cf, theta_norm = [], []
for lam in lambdas:
    theta = closed_form_ridge(X_s, y_c, lam)
    mse_train_cf.append(score_theta(X_s, y, theta, y_mean))
    theta_norm.append(np.linalg.norm(theta))

plt.figure()
plt.semilogx(lambdas, mse_train_cf, marker='o')
plt.title("Closed-form Ridge: Train MSE vs λ")
plt.xlabel("λ");
plt.ylabel("Train MSE");
plt.grid();
plt.show()
plt.figure()
plt.semilogx(lambdas, theta_norm, marker='o')
plt.title("Closed-form Ridge: ||θ||₂ vs λ")
plt.xlabel("λ");
plt.ylabel("Parameter norm");
plt.grid();
plt.show()
```



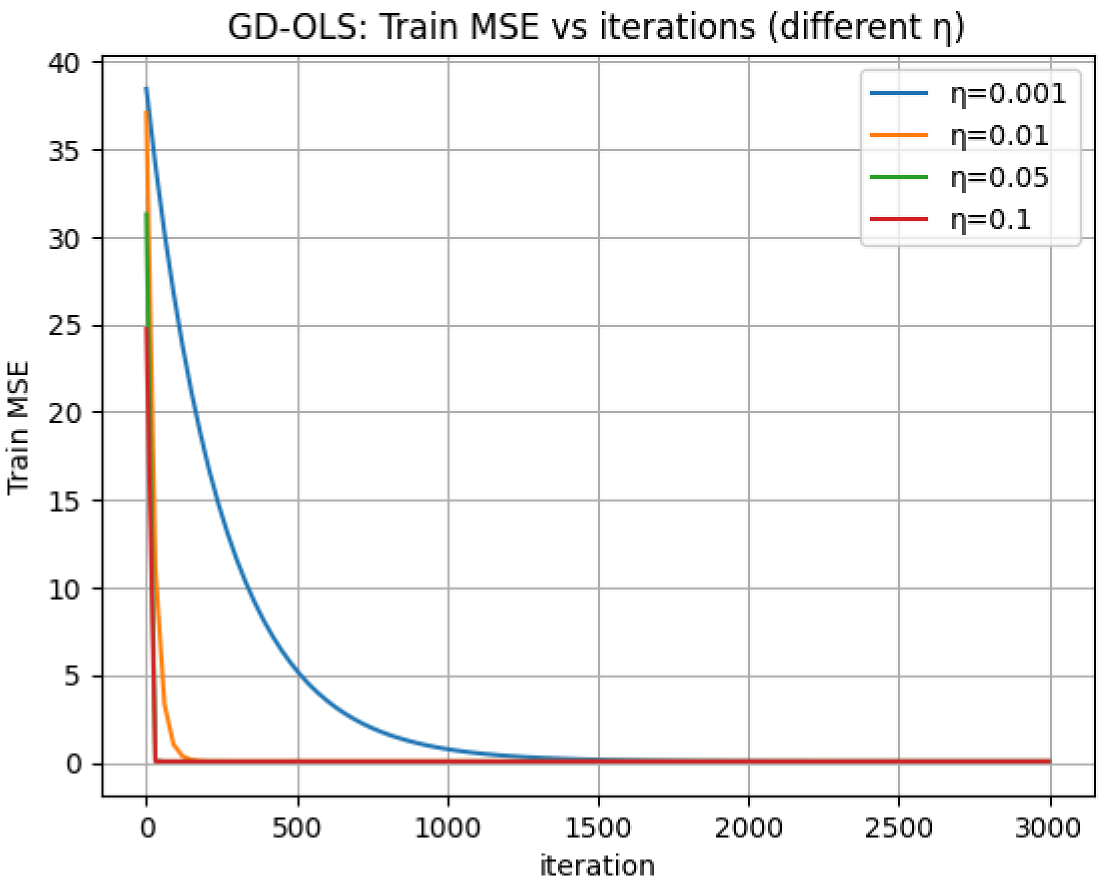



## Exercise 4

We implement vanilla gradient descent for OLS using the derived gradients. We track convergence for different learning rates and iteration counts.

In [ ]:
```
def gd_ols(Xs, y_c, eta=0.1, num_iters=1000, theta0=None, track=False):
    n, p = Xs.shape
    theta = np.zeros(p) if theta0 is None else theta0.astype(float).copy()
    hist = []
    for t in range(num_iters):
        grad = (2.0/n) * (Xs.T @ (Xs @ theta) - Xs.T @ y_c)
        theta -= eta * grad
        if track and (t % max(1, num_iters//100) == 0 or t == num_iters-1):
            mse = score_theta(Xs, y_c + y_mean, theta, y_mean)
            hist.append((t, mse, np.linalg.norm(grad)))
    return (theta, hist) if track else theta

# Comparing different learning rates
etas = [0.001, 0.01, 0.05, 0.1]
histories = {}
for eta in etas:
    theta_eta, hist = gd_ols(X_s, y_c, eta=eta, num_iters=3000, track=True)
    histories[eta] = hist

plt.figure()
for eta, hist in histories.items():
    iters = [h[0] for h in hist]
    mses  = [h[1] for h in hist]
    plt.plot(iters, mses, label=f"η={eta}")
```

```
plt.title("GD-OLS: Train MSE vs iterations (different η)")
plt.xlabel("iteration"); plt.ylabel("Train MSE"); plt.legend(); plt.grid(); plt.
```




**4b)** We implement Ridge GD with tolerance on parameter updates.
Stop when $\|\theta^{(t)} - \theta^{(t-1)}\|_2 < \mathrm{tol}$ or after num_iters.

In [ ]:
```
def gd_ridge(Xs, y_c, lam, eta=0.05, num_iters=5000, tol=1e-8, theta0=None, trac
    n, p = Xs.shape
    theta = np.zeros(p) if theta0 is None else theta0.astype(float).copy()
    hist = []
    for t in range(num_iters):
        grad = (2.0/n) * (Xs.T @ (Xs @ theta) - Xs.T @ y_c) + 2.0 * lam * theta
        theta_new = theta - eta * grad
        if track and (t % max(1, num_iters//100) == 0 or t == num_iters-1):
            mse = score_theta(Xs, y_c + y_mean, theta_new, y_mean)
            hist.append((t, mse, np.linalg.norm(grad)))
        if np.linalg.norm(theta_new - theta) < tol:
            theta = theta_new
            break
        theta = theta_new
    return (theta, hist) if track else theta

# Exploring effect of λ with fixed η
lam_list = [0.0, 1e-4, 1e-2, 1e-1]
gd_histories = {}
for lam in lam_list:
    theta_gd, hist = gd_ridge(X_s, y_c, lam=lam, eta=0.05, num_iters=5000, tol=1
    gd_histories[lam] = hist

plt.figure()
for lam, hist in gd_histories.items():
    iters = [h[0] for h in hist]
```

```
    mses  = [h[1] for h in hist]
    label = "OLS (λ=0)" if lam==0.0 else f"Ridge (λ={lam})"
    plt.plot(iters, mses, label=label)
plt.title("GD: Train MSE vs iterations (OLS vs Ridge)")
plt.xlabel("iteration"); plt.ylabel("Train MSE"); plt.legend(); plt.grid(); plt.
```


## Closed-form vs GD

We check that gradient descent converges to the closed-form solutions (within tolerance) for both **OLS** and **Ridge**.

In [10]:
```
def compare_solutions(Xs, y_c, lam, eta=0.05, num_iters=10000):
    if lam == 0.0:
        theta_cf = closed_form_ols(Xs, y_c)
        theta_gd = gd_ols(Xs, y_c, eta=eta, num_iters=num_iters)
    else:
        theta_cf = closed_form_ridge(Xs, y_c, lam)
        theta_gd = gd_ridge(Xs, y_c, lam=lam, eta=eta, num_iters=num_iters)
    diff = np.linalg.norm(theta_cf - theta_gd)
    return theta_cf, theta_gd, diff

for lam in [0.0, 1e-4, 1e-2, 1e-1]:
    cf, gd, d = compare_solutions(X_s, y_c, lam, eta=0.05, num_iters=15000)
    print(f"λ={lam:>6}: ||θ_cf - θ_gd||₂ = {d:.6e}")
```

```
λ=   0.0: ||θ_cf - θ_gd||₂ = 7.136584e-15
λ=0.0001: ||θ_cf - θ_gd||₂ = 6.145515e-04
λ=  0.01: ||θ_cf - θ_gd||₂ = 6.083856e-02
λ=   0.1: ||θ_cf - θ_gd||₂ = 5.581056e-01
```

# Exercise 5

We create a **sparse** linear model with 10 features, where only 3 are non-zero in the ground truth: $\theta_{true} = [5, -3, 0, 0, 0, 0, 2, 0, 0, 0]$. We compare OLS and Ridge (closed-form and GD) on this dataset.

In [11]:
```
# Generate sparse linear dataset
n_samples, n_features = 100, 10
theta_true = np.array([5.0, -3.0, 0.0, 0.0, 0.0, 0.0, 2.0, 0.0, 0.0, 0.0])
X_lin = np.random.normal(0, 1, size=(n_samples, n_features))
noise = 0.5 * np.random.normal(0, 1, size=n_samples)
y_lin = X_lin @ theta_true + noise

# Standardize X, center y
scaler_lin = StandardScaler().fit(X_lin)
X_lin_s = scaler_lin.transform(X_lin)
y_lin_mean = y_lin.mean()
y_lin_c = y_lin - y_lin_mean

# Closed-form
theta_ols_cf_lin = closed_form_ols(X_lin_s, y_lin_c)
theta_ridge_cf_lin = closed_form_ridge(X_lin_s, y_lin_c, lam=1e-2)

# Gradient descent
theta_ols_gd_lin = gd_ols(X_lin_s, y_lin_c, eta=0.05, num_iters=15000)
theta_ridge_gd_lin = gd_ridge(X_lin_s, y_lin_c, lam=1e-2, eta=0.05, num_iters=15

def summarize_fit(name, theta_est):
    mse_tr = score_theta(X_lin_s, y_lin, theta_est, y_lin_mean)
    print(f"{name:20s} | Train MSE: {mse_tr:8.4f} | ||θ - θ_true||₂: {np.linalg.

print("== Sparse linear regression ==")
summarize_fit("OLS (closed-form)", theta_ols_cf_lin)
summarize_fit("Ridge (closed-form)", theta_ridge_cf_lin)
summarize_fit("OLS (GD)", theta_ols_gd_lin)
summarize_fit("Ridge (GD)", theta_ridge_gd_lin)
```

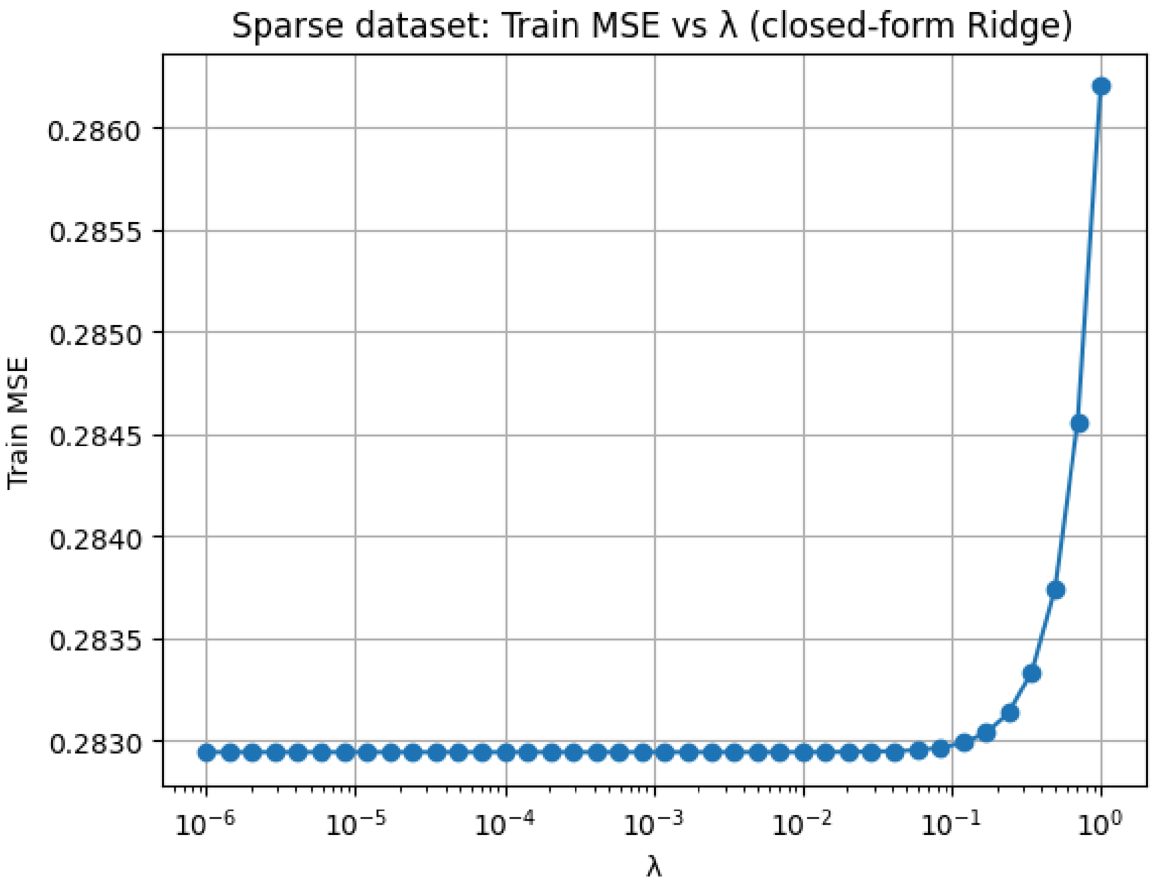
```
== Sparse linear regression ==
OLS (closed-form)    | Train MSE:   0.2829 | ||θ - θ_true||₂:   0.5161
Ridge (closed-form)  | Train MSE:   0.2829 | ||θ - θ_true||₂:   0.5164
OLS (GD)             | Train MSE:   0.2829 | ||θ - θ_true||₂:   0.5161
Ridge (GD)           | Train MSE:   0.2862 | ||θ - θ_true||₂:   0.5482
```

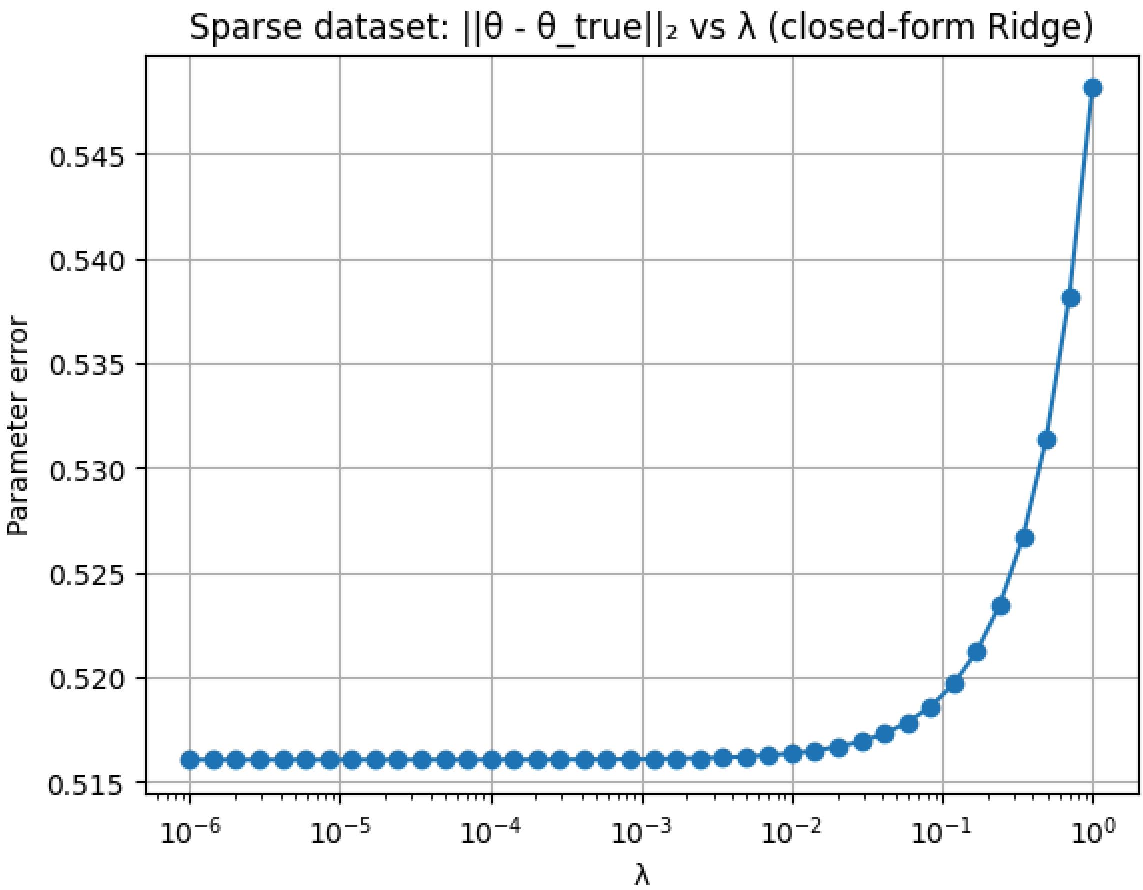
## Which method performs best?

Because the true model is sparse and features are standardized, Ridge typically stabilizes estimates in the presence of noise and collinearity. We therefore expect Ridge to achieve a lower parameter error $||\hat{\theta} - \theta_{true}||_2$ and often competitive or lower MSE, depending on $\lambda$. You can sweep $\lambda$ to observe the bias–variance trade-off.

In [12]:
```
lambdas_lin = np.logspace(-6, 0, 40)
mse_lin_cf, param_err_cf = [], []
for lam in lambdas_lin:
    th = closed_form_ridge(X_lin_s, y_lin_c, lam)
    mse_lin_cf.append(score_theta(X_lin_s, y_lin, th, y_lin_mean))
```

```
    param_err_cf.append(np.linalg.norm(th - theta_true))

plt.figure()
plt.semilogx(lambdas_lin, mse_lin_cf, marker='o')
plt.title("Sparse dataset: Train MSE vs λ (closed-form Ridge)")
plt.xlabel("λ"); plt.ylabel("Train MSE"); plt.grid(); plt.show()

plt.figure()
plt.semilogx(lambdas_lin, param_err_cf, marker='o')
plt.title("Sparse dataset: ||θ - θ_true||₂ vs λ (closed-form Ridge)")
plt.xlabel("λ"); plt.ylabel("Parameter error"); plt.grid(); plt.show()
```






## Takeaways

- Scaling features and centering the target simplifies intercept handling and stabilizes optimization.
- Closed-form OLS/Ridge agree with gradient descent when step size and iterations are appropriate.
- Learning rate controls convergence speed/stability; if set too big can diverge, too small converges slowly.
- Ridge (λ) shrinks coefficients, trading bias for variance reduction; helpful in noisy or ill-conditioned settings.