M+E+C:
Nonlinear Optimization

Keith O'Hara

01/19/2018

Optimization

- General formulation of the unconstrained problem:

$$\min_{\mathbf{x} \in \mathcal{X}} f(\mathbf{x})$$

- See Nocedal and Wright (2006) for details.
- Separate problems into
  (i) convex optimization
  (ii) non-convex optimization
- **Theorem:** *(N-W, 2.5)* When f is convex, any local minimizer x^* is a global minimizer of f. If in addition f is differentiable, then any stationary point x^* is a global minimizer of f.
- For non-convex problems:
  (i) derivative vs derivative-free methods
  (ii) rates of convergence

Newton Methods

- Taylor's theorem (with Lagrange's form for the remainder):

$$f(\mathbf{x}+\mathbf{d}) = f(\mathbf{x}) + \mathbf{d}'\nabla f(\mathbf{x}+\mathbf{d}) + \frac{1}{2}\mathbf{d}'[\nabla^2 f(\mathbf{x}+t\mathbf{d})]\mathbf{d} \qquad (1)$$

  for some $t \in [0,1]$.
- Then

$$f(\mathbf{x}_k+\mathbf{d}) \approx f(\mathbf{x}_k) + \mathbf{d}'\nabla f_k + \frac{1}{2}\mathbf{d}'[\nabla^2 f_k]\mathbf{d} =: \mathbf{m}_k(\mathbf{d}) \qquad (2)$$

- Setting the derivative equal to zero, we can solve for the descent direction $\mathbf{d}$:

$$\mathbf{d} = -[\nabla^2 f_k]^{-1}\nabla f_k$$

- The Newton direction is reliable when the difference $|f(\mathbf{x}_k+\mathbf{d}) - \mathbf{m}_k(\mathbf{d})|$ is not too large.

Line Search

- There are really two steps in convex minimization:
  1. Find the descent direction $\mathbf{d}$
  2. Decide how far along this direction we should go
- That is, for a direction $\mathbf{d}$, search along $\mathbf{d}$ from the current iterate $\mathbf{x}_k$ for a new iterate with a lower function value.
- The distance to move along $\mathbf{d}$ can be found by approximately solving the following one-dimensional minimization problem:

$$\alpha_k := \underset{\alpha}{\operatorname{argmin}}\, f(\mathbf{x}_k + \alpha \mathbf{d}) \tag{3}$$

- By solving (3) exactly, we would derive the maximum benefit from the direction $\mathbf{d}$, but an exact minimization may be expensive and is usually unnecessary.

Inexact Line Search

- Instead we use 'inexact' methods.
- **Strong Wolfe conditions**:

$$f(\mathbf{x}_k + \alpha_k \mathbf{d}_k) \leq f(\mathbf{x}) + c_1 \alpha_k \mathbf{d}_k' \nabla f_k \tag{SD}$$

$$|\mathbf{d}_k' \nabla f(\mathbf{x}_k + \alpha_k \mathbf{d}_k)| \leq c_2 |\mathbf{d}_k' \nabla f_k| \tag{Curv}$$

  where $0 < c_1, c_2, < 1$ are tuning parameters.
- The Wolfe conditions are invariant to multiplication by a constant or an affine change of variables.
- Methods like this are important to guarantee global convergence of quasi-Newton methods
  - Under regularity conditions, Newton's method will convergence at a quadratic rate. Quasi-Newton will converge super-linearly.
- More and Thuente (1994)

Quasi-Newton: BFGS

- For the descent direction, replace $[\nabla^2 f_k]^{-1}$ by an approximation W_i:

$$\mathbf{d}_i = -W_i \nabla f_i$$

- Update formula:

$$\begin{aligned} s &:= x_{i+1} - x_i \\ y &:= \nabla f_{i+1} - \nabla f_i \\ W_{i+1} &:= \left[I - \frac{sy'}{y's}\right] W_i \left[I - \frac{sy'}{y's}\right]' + \frac{ss'}{y's} \end{aligned}$$

- Update

$$x_{i+1} = x_i + \alpha_i d_i$$

where

$$\alpha_i := \underset{\alpha}{\operatorname{argmin}}\, f(x_i + \alpha d_i)$$

Conjugate Gradient

- The descent direction:

$$d_i = -\nabla f_i + \beta_i d_{i-1}$$

  The value β_i 'ensures that d_i and d_{i-1} are **conjugate**.'

- Update

$$x_{i+1} = x_i + \alpha_i d_i$$

  where

$$\alpha_i := \underset{\alpha}{\operatorname{argmin}}\, f(x_i + \alpha d_i)$$

- Two common choices for β:

$$\beta_{\mathrm{FR}} = \frac{\nabla f_i \cdot \nabla f_i}{\nabla f_{i-1} \cdot \nabla f_{i-1}} \tag{4}$$

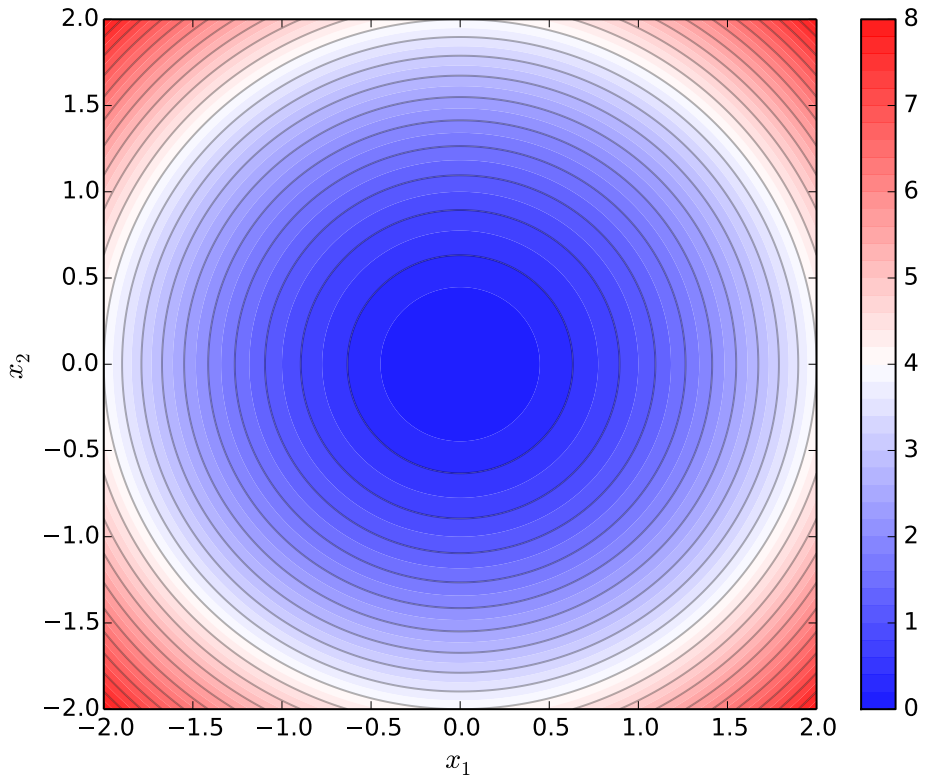
$$\beta_{\mathrm{PR+}} = \max\left\{0, \frac{\nabla f_i \cdot (\nabla f_i - \nabla f_{i-1})}{\nabla f_{i-1} \cdot \nabla f_{i-1}}\right\} \tag{5}$$

Example: Sphere Function

- Minimize

$$\min_{\mathbf{x}} \left\{ \sum_{k=1}^{n} x_k^2 \right\}$$

- Note: $\nabla f(\mathbf{x}) = 2\mathbf{x}$. In two dimensions:



Example: Sphere Function

R code:

```
fn <- function(x) { return(sum(x*x)) }
gn <- function(x) { return(2*x) }
n <- 10
x0 <- rep(2,n)
optim(x0,fn,gn,method="BFGS")
```

Example: Logistic Regression

- Data:

$$y_i|\mathbf{x}_i, \boldsymbol{\theta} \sim \text{Bern}(\mu_i)$$

  where $\mu_i := \text{sigm}(\boldsymbol{\theta}'\mathbf{x}_i)$, $\text{sigm}(x) = \frac{1}{1+\exp(-x)}$.
- Objective function:

$$f(\boldsymbol{\theta}) := -\sum_{i=1}^{N} [y_i \ln(\mu_i) + (1 - y_i) \ln(1 - \mu_i)]$$

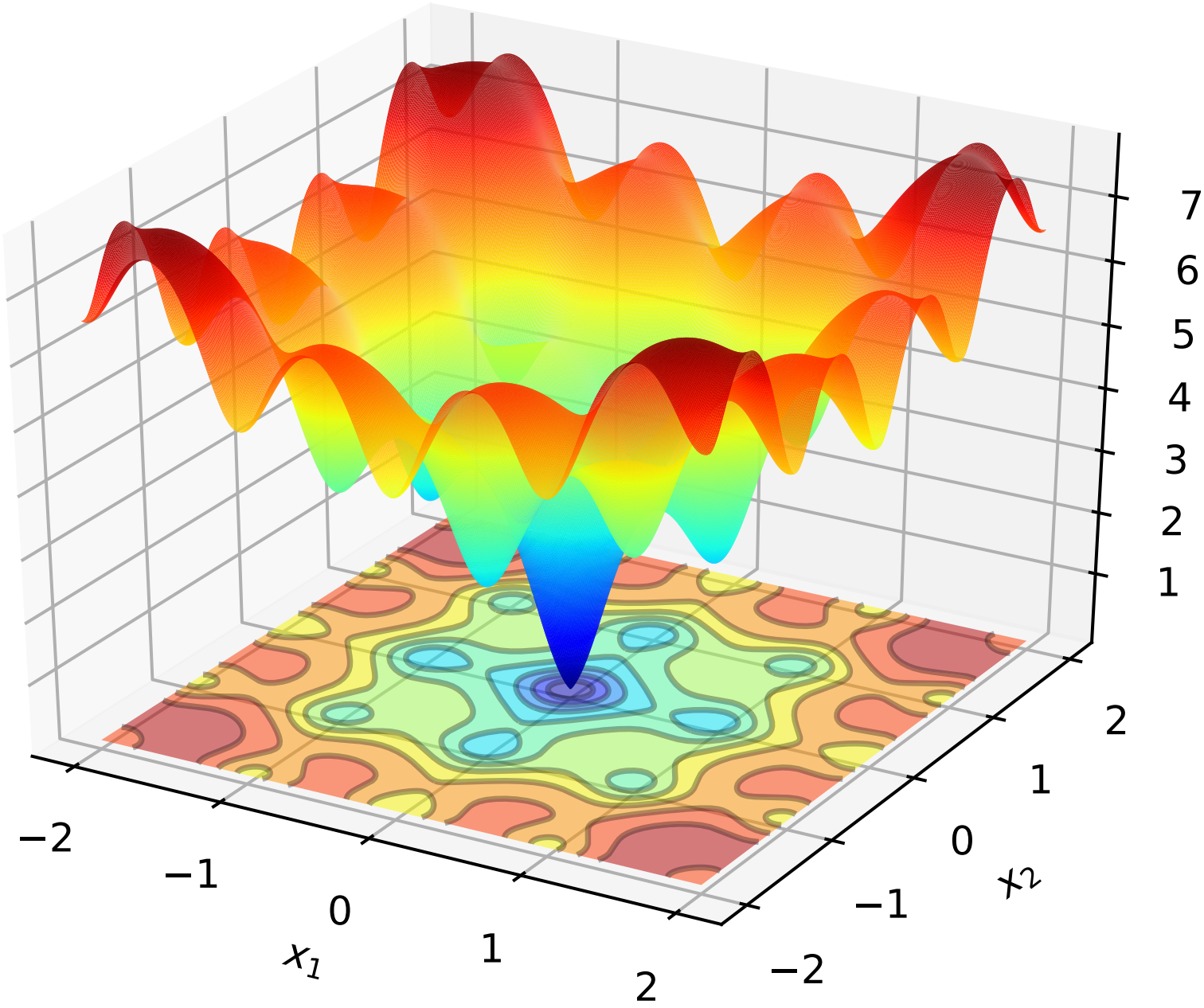
- Gradient and Hessian:

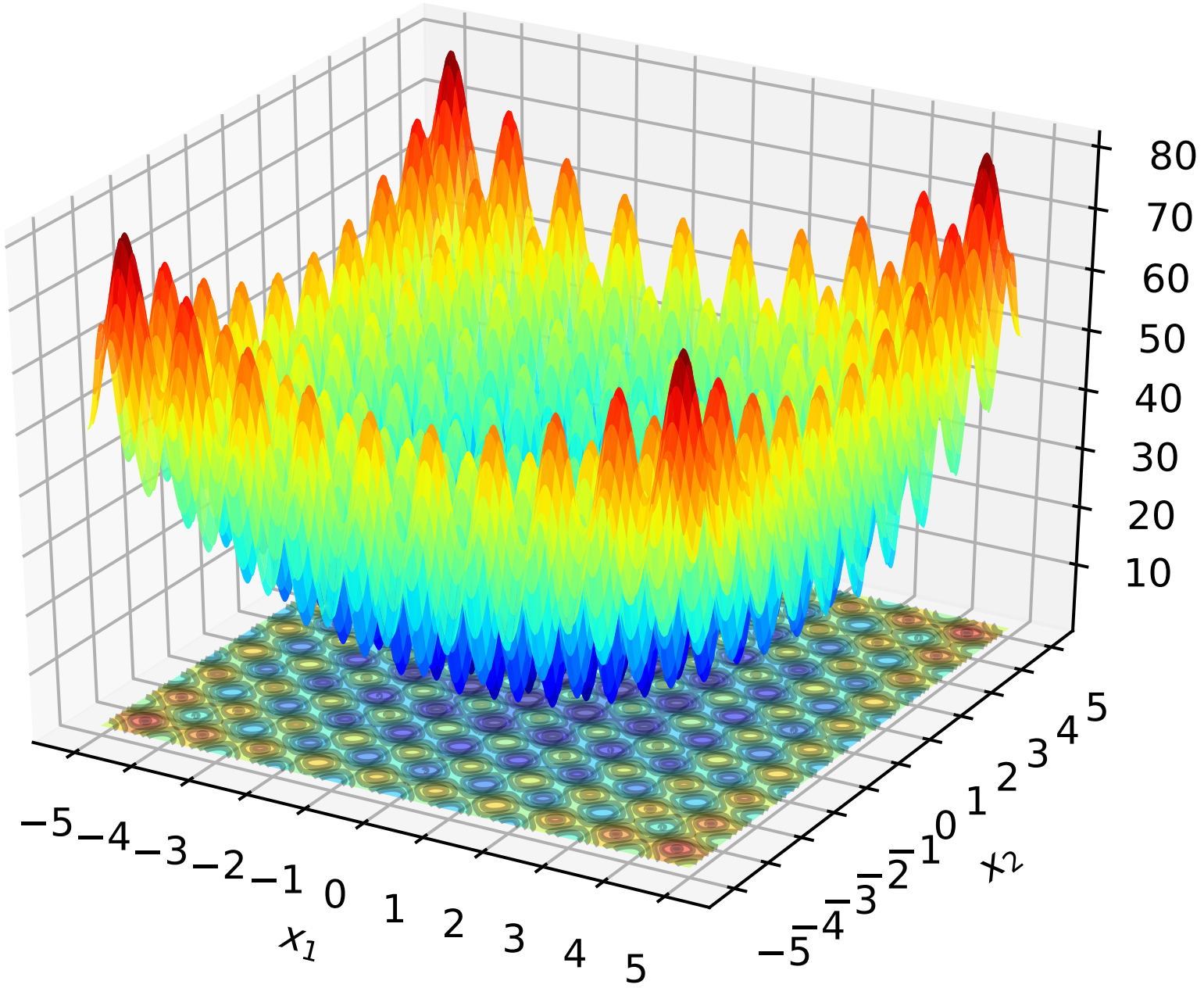
$$\nabla_{\boldsymbol{\theta}} f := \mathbf{X}'(\boldsymbol{\mu} - \mathbf{y})$$

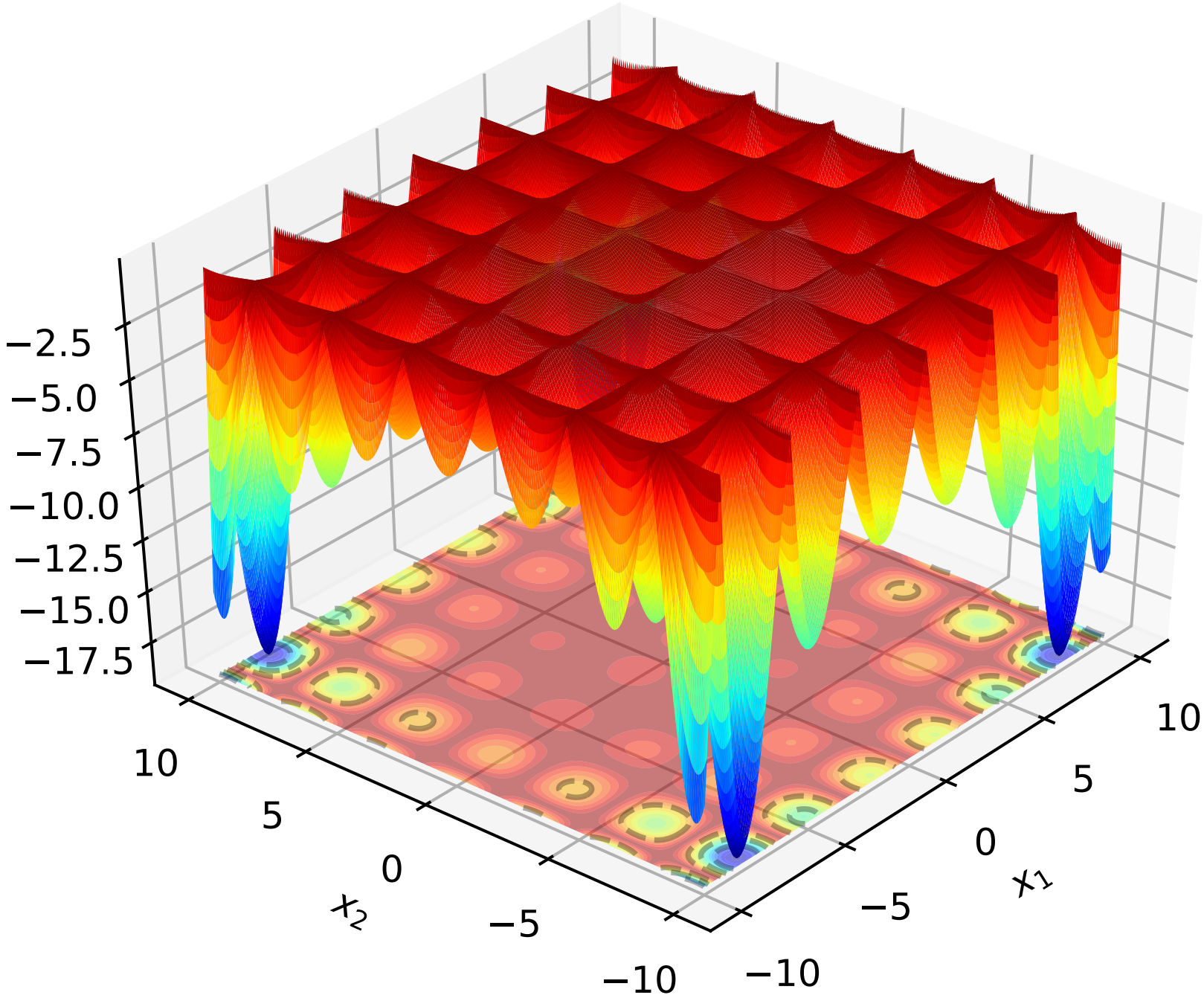
$$\mathbf{H} := \nabla_{\boldsymbol{\theta}}[\nabla_{\boldsymbol{\theta}} f]' = \mathbf{X}'\mathbf{S}\mathbf{X}$$

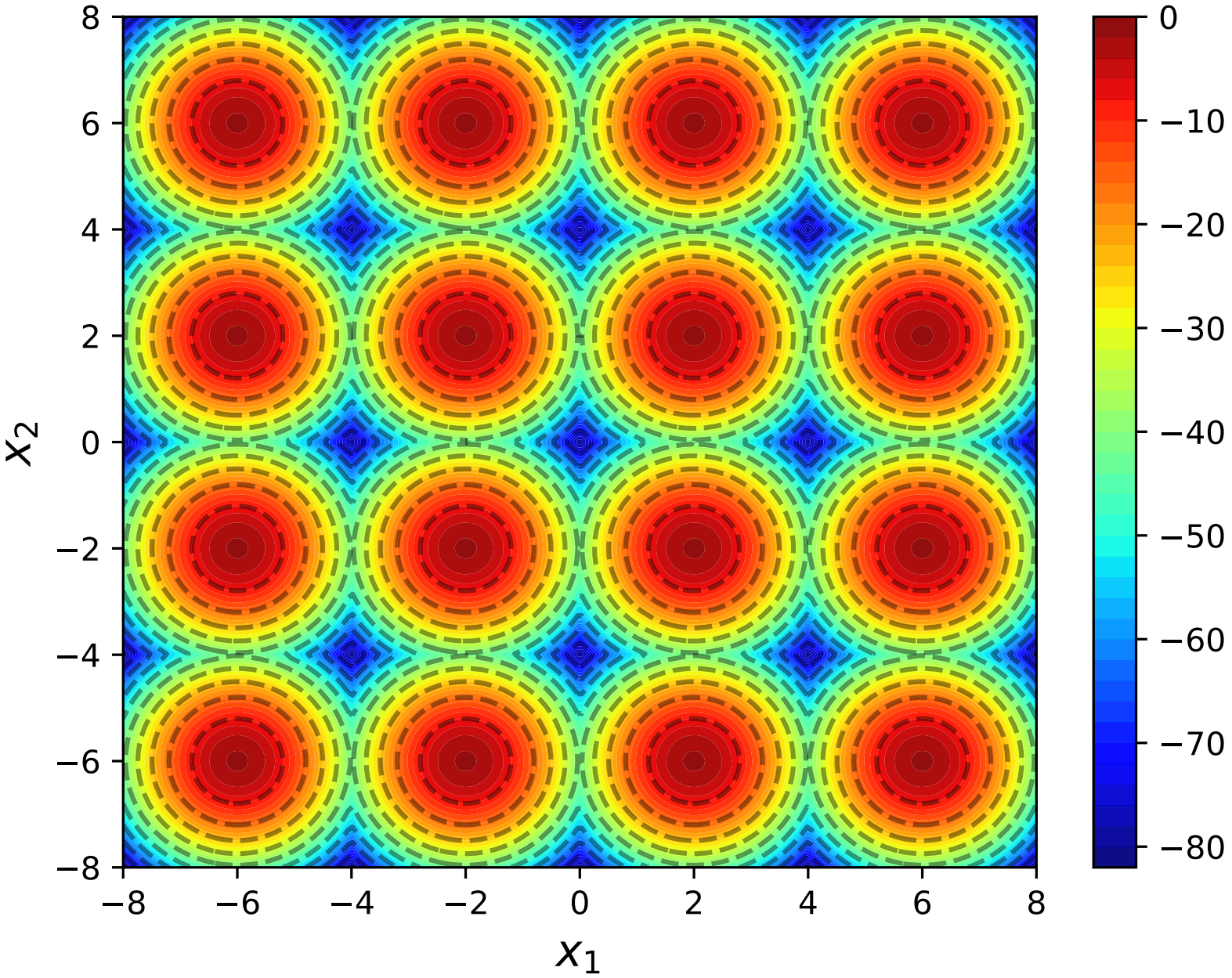
  $\mathbf{S} := \text{diag}(\mu_i(1 - \mu_i))$.

Nonconvex Problems


7
6
5
4
3
2
1
−2
−1
0
1
2
x_1
2
1
0
−1
−2
x_2


80
70
60
50
40
30
20
10
−5 −4 −3 −2 −1 0 1 2 3 4 5
x_1
−5 −4 −3 −2 −1 0 1 2 3 4 5
x_2


−2.5
−5.0
−7.5
−10.0
−12.5
−15.0
−17.5
10
5
0
−5
−10
x_2
10
5
0
−5
−10
x_1


8
6
4
2
0
−2
−4
−6
−8
x_2
−8
−6
−4
−2
0
2
4
6
8
x_1
0
−10
−20
−30
−40
−50
−60
−70
−80

Differential Evolution

- Let X_i denote a $N_p \times d$-dimensional array of values at stage i of the algorithm.
- Three steps.

1. **Mutation step.** For random and unique indices a, b, c

$$X^* = X_i(c, :) + F \times (X_i(a, :) - X_i(b, :))$$

2. **The Crossover Step.** For a random integer $r_k \in \{1, \ldots, d\}$,

$$X_c^*(j,k) = \begin{cases} X^*(j,k) & \text{if } u_k \leq CR \text{ or } k = r_k \\ X_i(j,k) & \text{else} \end{cases}$$

2. **The Update (or Selection) Step.**

$$X_{i+1}(j,:) = \begin{cases} X_c^*(j,:) & \text{if } f(X_c^*(j)) < f(X_i(j)) \\ X_i(j) & \text{else} \end{cases}$$

Example: Ackley's Function

$$\min_{x\in[-5,5]^2}\left\{-20\exp\left(-0.2\sqrt{0.5(x_1^2+x_2^2)}\right)-\exp\left(0.5[\cos(2\pi x_1)+\cos(2\pi x_2)]\right)+e+20\right\}$$

```
double
ackley_fn(const arma::vec& vals_inp, arma::vec* grad_out, void* opt_data)
{
    const double x = vals_inp(0), y = vals_inp(1);
    const double pi = arma::datum::pi;

    double obj_val = -20*std::exp( -0.2*std::sqrt(0.5*(x*x + y*y)) ) \
    - std::exp( 0.5*(std::cos(2*pi*x) + std::cos(2*pi*y)) ) + 22.718282L;
    //
    return obj_val;
}

int main() {
    ...
    bool success = optim::de(x,ackley_fn,nullptr);
    ...
}
```

```
de: Ackley test completed successfully.
elapsed time: 0.028167s

de: solution to Ackley test:
  -1.2702e-17
  -3.8432e-16
```