1. Lorenz 63

考虑洛伦兹63系统

$$\frac{dx_1}{dt} = \sigma(x_2 - x_1),$$
$$\frac{dx_2}{dt} = x_1(r - x_3) - x_2,$$
$$\frac{dx_3}{dt} = x_1 x_2 - \beta x_3;$$

这些参数满足 $\sigma, r, \beta \in \mathcal{R}^+$。当我们取 $\sigma = 10, \beta = \frac{8}{3}$ 时，我们有

- 当$r < 24.74$，系统不是混沌的，会收敛到一个吸引子
- 当$24.74 < r < 148.4$， 系统变得混沌

In [86]:

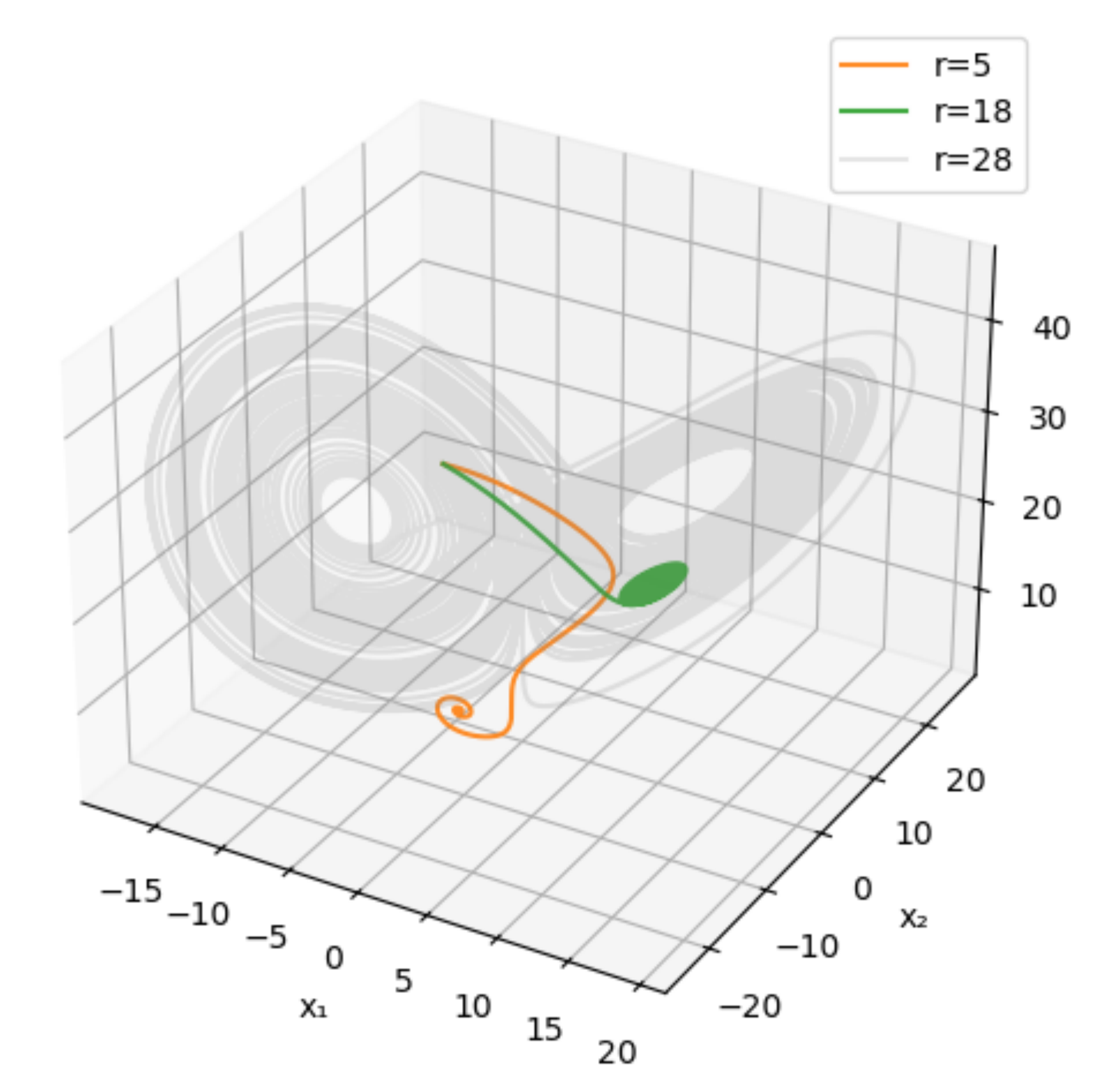
```
using PyPlot
using Statistics
using LinearAlgebra


function f(x::Array{FT,1}, σ::FT, r::FT, β::FT) where {FT<:AbstractFloat}
    return [σ*(x[2]-x[1]); x[1]*(r-x[3])-x[2]; x[1]*x[2]-β*x[3]]
end

function df_dx(x::Array{FT,1}, σ::FT, r::FT, β::FT) where {FT<:AbstractFloat}
    return  [-σ         σ        0.0;
             (r-x[3])   -1.0     -x[1];
             x[2]       x[1]     -β]
end

function df_dθ(x::Array{FT,1}, σ::FT, r::FT, β::FT) where {FT<:AbstractFloat}
    return [(x[2]-x[1]) 0.0      0.0;
            0.0         x[1]     0.0;
            0.0          0.0     -x[3]]
end

# integrate Lorenz63 with forward Euler method
function compute_Lorenz63_FE(x0::Array{FT,1}, θ::Array{FT,1}, Δt::FT, N_t::IT) where {FT<:AbstractFloat, IT<:In

    N_x = length(x0)

    σ, r, β = θ

    xs = zeros(N_x, N_t+1)
    xs[:, 1] = x0
    for i = 1:N_t
        xs[:, i+1] = xs[:, i] + Δt*f(xs[:, i], σ, r, β)
    end

    return xs
end


function f_mean_x3(x::Array{FT,1}) where {FT<:AbstractFloat}
    J = [x[3];]
    pJ_px = FT.([0 0 1])
    return J, pJ_px
end


# Compute objective function J, and dJ/dxs
function compute_J(xs::Array{FT,2}, func::Function, j::IT, k::IT) where {FT<:AbstractFloat, IT<:Int}
    # J is defined as the average
    # J = ∑_{i=j}^{k} func(x(t_i)) /(k-j+1)
    # return J and dJ/dxs
    N_x, N_t = size(xs, 1), size(xs, 2) - 1
    J = 0.0
    ∂J_∂xs = zeros(N_x, N_t+1)

    for i = j:k
        J_i, ∂J_∂xs_i = func(xs[:,i])
        J +=  J_i[1]
        ∂J_∂xs[:,i] = ∂J_∂xs_i
    end

    ∂J_∂xs ./= (k-j+1)
    J /= (k-j+1)
    return J, ∂J_∂xs
end


# Compute objective function dJ/dθ
function compute_gradient_adjoint(θ::Array{FT,1}, xs::Array{FT,2}, ∂J_∂xs::Array{FT,2}, Δt::FT) where {FT<:Abstr
    N_x, N_t = size(xs, 1), size(xs, 2) - 1
    σ, r, β = θ
    N_θ = length(θ)
    λs = zeros(N_x, N_t+1)
    λs[:,N_t+1] .= ∂J_∂xs[:,N_t+1]

    for i=N_t:-1:1
        λs[:,i] = ∂J_∂xs[:,i] + λs[:,i+1] + Δt*df_dx(xs[:,i], σ, r, β)'*λs[:,i+1]
    end

    dJ_dθ = zeros(FT, N_θ)
    for i=2:N_t+1
        dJ_dθ .+= df_dθ(xs[:,i-1], σ, r, β)' * λs[:,i]*Δt
    end

    return dJ_dθ
end


function compute_dx3_dr_adjoint(rs = Array(LinRange(0, 50, 1001)))
    FT = Float64
    Tobs = 20.0
    Tspinup = 30.0
    Δt = 0.001

    T = Tobs + Tspinup
    N_t = Int64(T/Δt)
    N_burn_in = Int64(Tspinup/Δt)

    N_r = length(rs)

    mean_x3s = zeros(FT, N_r)
    dmean_x3_drs = zeros(FT, N_r)

    x0 = [-8.0; 5.0; 25]
    for i = 1:N_r
        θ = [10.0; rs[i]; 8.0/3.0]
        xs = compute_Lorenz63_FE(x0, θ, Δt, N_t)

        J, ∂J_∂xs = compute_J(xs, f_mean_x3,  N_burn_in + 1, N_t + 1)
        dJ_dθ = compute_gradient_adjoint(θ, xs, ∂J_∂xs, Δt)
        mean_x3s[i], dmean_x3_drs[i] = J, dJ_dθ[2]
    end

    return  rs, mean_x3s, dmean_x3_drs
end
```

```
WARNING: method definition for compute_gradient_adjoint at In[86]:68 declares type variable IT but does not use it.
```

Out[86]: compute_dx3_dr_adjoint (generic function with 2 methods)

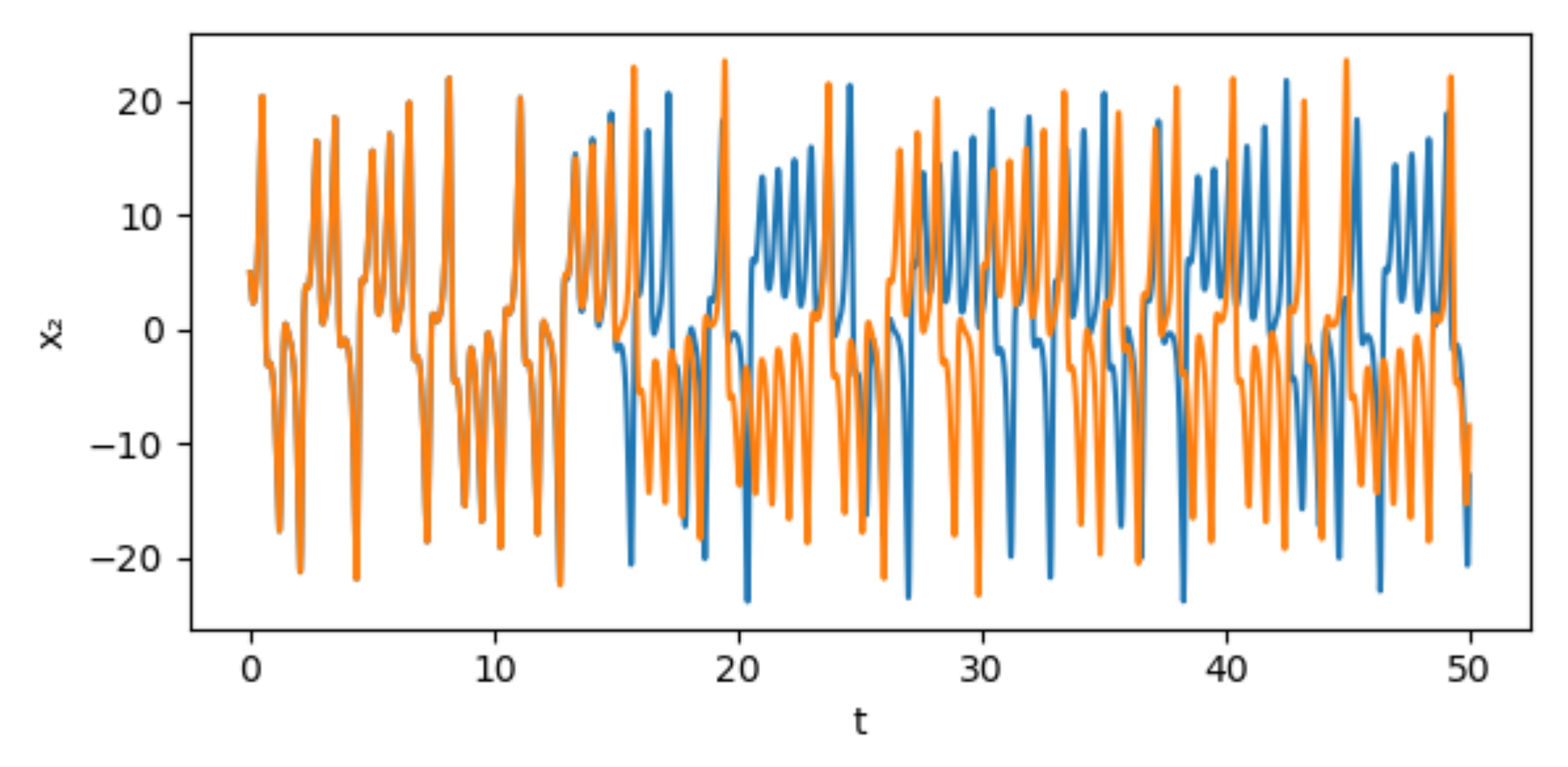
In [87]:

```
T = 100
Δt = 0.001
N_t = Int64(T/Δt)
x0 = [-8.0; 5.0; 25]

fig = plt.figure()
ax = fig.add_subplot(projection="3d")



θ = [10.0; 5.0; 8.0/3.0]
xs = compute_Lorenz63_FE(x0, θ, Δt, N_t)
ax.plot(xs[1,:], xs[2,:], xs[3,:], color="C1", label="r=5", alpha=0.9)
ax.legend()


θ = [10.0; 18.0; 8.0/3.0]
xs = compute_Lorenz63_FE(x0, θ, Δt, N_t)
ax.plot(xs[1,:], xs[2,:], xs[3,:], color="C2", label="r=18", alpha=0.9)
ax.legend()


θ = [10.0; 28.0; 8.0/3.0]
xs = compute_Lorenz63_FE(x0, θ, Δt, N_t)
ax.plot(xs[1,:], xs[2,:], xs[3,:], color="grey", label="r=28", alpha=0.2)
ax.legend()

ax.set_xlabel("x₁")
ax.set_ylabel("x₂")
ax.set_zlabel("x₃")

fig.tight_layout()
fig.savefig("Lorenz63-r.pdf")
```



In [90]:

```
θ = [10.0; 28.0; 8.0/3.0]
T = 50
Δt = 0.001
N_t = Int64(T/Δt)
ts = Array(LinRange(0, T, N_t+1))


x0 = [-8.0; 5.0; 25]
xs = compute_Lorenz63_FE(x0, θ, Δt, N_t)
fig= PyPlot.figure(figsize=(6,3))
PyPlot.plot(ts, xs[2,:])


x0 = [-8.0; 5.0; 25] + [0; 1e-6; 0]
xs = compute_Lorenz63_FE(x0, θ, Δt, N_t)
PyPlot.plot(ts, xs[2,:])

PyPlot.xlabel("t")
PyPlot.ylabel("x₂")
PyPlot.tight_layout()
fig.savefig("Lorenz63-t.pdf")
```



离散伴随法

我们定义

$$\mathcal{G}(r) = \frac{1}{20}\int_{30}^{50} q\big(x_3(t)\big)dt.$$

伴随法是用来计算导数 $\frac{d\mathcal{G}(r)}{dr}$ 的方法，考虑以下常微分方程约束的优化问题

$$\mathcal{G}(\theta) = \frac{1}{T}\int_0^T q(x,\theta,t) \quad \text{subject to} \quad \frac{dx}{dt} = f(x,\theta) \qquad x(0) = x_0(\theta)$$

离散伴随法首先离散常微分方程，然后写出伴随方程计算导数；与之对应的是连续伴随法，它首先写出连续的伴随方程，再离散求导数。 离散伴随法能给出精确的导数，但是依赖方程的离散格式。我们采用向前欧拉方法对常微分方程进行离散

$$\begin{aligned} J = \frac{1}{N_t - N_s}\sum_{n=N_s}^{N_t-1} q(x^n,\theta) \\ \text{subject to} \quad & x^0 = x_0(\theta) \\ & x^n = x^{n-1} + \Delta t f(x^{n-1},\theta,t^{n-1}) \quad n = 1,2,\cdots N_t \end{aligned}$$

我们可以把它当成优化问题，采用拉格朗日乘子法定义，定义拉格朗日乘子

$$L = J - \sum_{n=0}^{N_t} \lambda^{n^T} \tilde{f}^n,$$

其中

$$\begin{aligned} &\tilde{f}^0(x^0,\theta) = x^0 - x_0(\theta) \\ &\tilde{f}^n(x^n,x^{n-1},\theta) = x^n - x^{n-1} - \Delta t f(x^{n-1},\theta,t^{n-1}) \quad n = 1,2,\cdots N_t \end{aligned}$$

我们写出拉格朗日乘子的全导数

$$\begin{aligned} \frac{dL}{d\theta} =& \frac{\partial J}{\partial \theta} + \sum_{n=0}^{N_t} \frac{\partial J}{\partial x^n}\frac{\partial x^n}{\partial \theta} \\ & - \sum_{n=1}^{N_t} \lambda^{n^T}\big(\frac{\partial \tilde{f}^n}{\partial x^n}\frac{\partial x^n}{\partial \theta} + \frac{\partial \tilde{f}^n}{\partial x^{n-1}}\frac{\partial x^{n-1}}{\partial \theta} + \frac{\partial \tilde{f}^n}{\partial \theta}\big) \\ & - \lambda^{0^T}\big(\frac{\partial \tilde{f}^0}{\partial x^0}\frac{\partial x^0}{\partial \theta} + \frac{\partial \tilde{f}^0}{\partial \theta}\big) \\ =& \frac{\partial J}{\partial \theta} - \sum_{n=0}^{N_t} \lambda^{n^T}\frac{\partial \tilde{f}^n}{\partial \theta} \\ & + \sum_{n=0}^{N_t-1} \frac{\partial x^n}{\partial \theta}\big(\frac{\partial J}{\partial x^n} - \lambda^{n^T}\frac{\partial \tilde{f}^n}{\partial x^n} - \lambda^{n+1^T}\frac{\partial \tilde{f}^{n+1}}{\partial x^n}\big) \\ & + \frac{\partial x^{N_t}}{\partial \theta}\big(\frac{\partial J}{\partial x^{N_t}} - \lambda^{N_t^T}\frac{\partial \tilde{f}^{N_t}}{\partial x^{N_t}}\big) \end{aligned}$$

我们可以消去 $\frac{\partial x^n}{\partial \theta}$，得到离散伴随方程：

$$\begin{aligned} \frac{\partial J}{\partial x^{N_t}} - \lambda^{N_t^T}\frac{\partial \tilde{f}^{N_t}}{\partial x^{N_t}} = 0 \\ \frac{\partial J}{\partial x^n} - \lambda^{n^T}\frac{\partial \tilde{f}^n}{\partial x^n} - \lambda^{n+1^T}\frac{\partial \tilde{f}^{n+1}}{\partial x^n} = 0 \qquad n = 0,1,\cdots N_t - 1 \end{aligned}$$

把 $\tilde{f}^n$ 的定义带入就能得到

$$\begin{aligned} \frac{\partial J}{\partial x^{N_t}} - \lambda^{N_t^T} = 0 \\ \frac{\partial J}{\partial x^n} - \lambda^{n^T} - \lambda^{n+1^T}\big(-I - \Delta t\frac{\partial f(x^n,\theta,t^n)}{\partial x^n}\big) = 0 \qquad n = 0,1,\cdots N_t - 1 \end{aligned}$$

最后的导数是

$$\begin{aligned} \frac{dJ}{d\theta} &= \frac{\partial J}{\partial \theta} - \sum_{n=0}^{N_t} \lambda^{n^T}\frac{\partial \tilde{f}^n}{\partial \theta} \\ &= \frac{\partial J}{\partial \theta} + \Delta t\sum_{n=1}^{N_t} \lambda^{n^T}\frac{\partial f(x^{n-1},\theta,t^{n-1})}{\partial \theta} + \lambda^{0^T}\frac{\partial x_0(\theta)}{\partial \theta} \end{aligned}$$

所以，导数的计算有两部，

- 正向求解常微分方程系统，储存状态量 $\{x^n\}$ 和目标函数 J
- 反向求解伴随方程，储存伴随量 $\{\lambda^n\}$ 总的计算量大概是求解常微分方程系统的两倍。

In [92]:

```
rs = Array(LinRange(0, 50, 1001))
rs, mean_x3s, dmean_x3_drs = compute_dx3_dr_adjoint(rs)
fig, (ax1, ax2) = PyPlot.subplots(ncols=2, figsize=(6,3))

ax1.plot(rs, mean_x3s, "--", fillstyle="none")
ax1.set_xlabel(L"r")
ax1.set_ylabel(L"\mathcal{G}(r)")
ax1.grid("on")



ax2.semilogy(rs, abs.(dmean_x3_drs), "or", markersize=1)
ax2.set_xlabel("\$r\$")
ax2.set_ylabel(L"|d\mathcal{G}(r)|")
ax2.grid("on")
fig.tight_layout()
fig.savefig("Lorenz63-g.png")
```