present

Line Search Stepsize Control and Trust-Region Methods

- Mathe 3 (CES)
- WS24/25
- Lambert Theisen (theisen@acom.rwth-aachen.de)

Stepsize Control Algorithm

backtracking_linesearch (generic function with 1 method)

```
function backtracking_linesearch(f, x, d, αmax, cond, β)
    @assert 0 < β < 1
    α = αmax
    while !cond(f, d, x, α)
        α *= β
    end
    @show α
    return α
end
```

Wolfe Stepsize Conditon

- We need to specify a conditon for the backtracking algorithm
- Use Wolfe conditions

$$\begin{aligned} \textbf{i)} \quad & f(\mathbf{x}_k + \alpha_k \mathbf{p}_k) \le f(\mathbf{x}_k) + c_1 \alpha_k \mathbf{p}_k^{\mathrm{T}} \nabla f(\mathbf{x}_k), \\ \textbf{ii)} \quad & -\mathbf{p}_k^{\mathrm{T}} \nabla f(\mathbf{x}_k + \alpha_k \mathbf{p}_k) \le -c_2 \mathbf{p}_k^{\mathrm{T}} \nabla f(\mathbf{x}_k), \end{aligned}$$

armijo (generic function with 1 method)

```
armijo(f, d, x, α) = f(x + α*d) <= f(x) + 1E-4 * α * derivative(f, x)' * d
```

```
curvature (generic function with 1 method)
```

```
curvature(f, d, x, α) = derivative(f, x + α*d)' * d >= 0.9 * derivative(f, x)' * d
```

```
backtracking_linesearch_wolfe (generic function with 1 method)
```

```
function backtracking_linesearch_wolfe(f, x, d, αmax, β)  #TODO
    return backtracking_linesearch(f, x, d, αmax, (f, d, x, α)->(armijo(f, d, x,
    α)&&curvature(f, d, x, α)), β)
end
```

Use Backtracking Algorithm in Gradient Descent

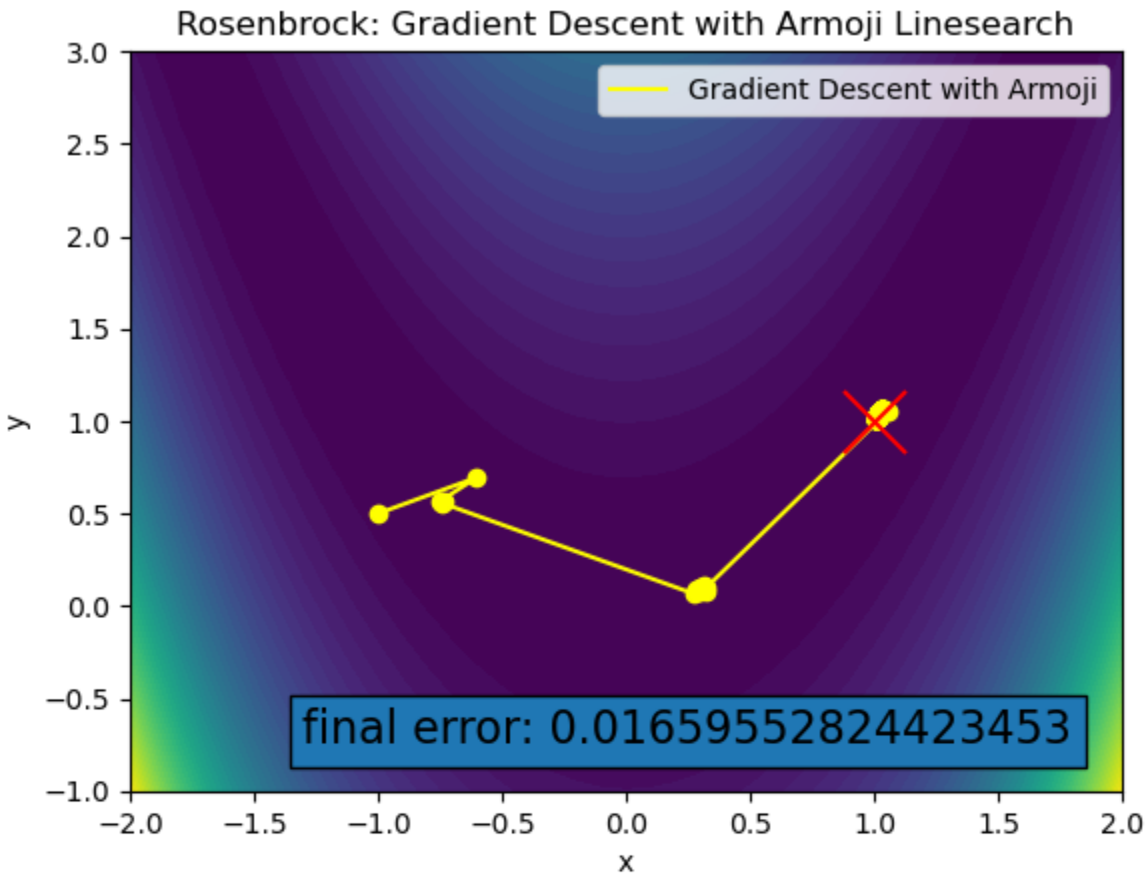
- Same as last week, but with adaptive step size

```
gradient_descent_wolfe (generic function with 1 method)
```

```
function gradient_descent_wolfe(f, x0, kmax)
    x = x0
    hist = []
    push!(hist, x)
    for k=1:kmax
        x = x + backtracking_linesearch_wolfe(
            f, x, -derivative(f, x), 2, 0.5
        ) * -derivative(f, x)
        push!(hist, x)
    end
    return x, hist
end
```

Rosenbrock: GD with Armijo

- Remember from last week: GD was very sensitive to step width
    - Even divergence for most stepsizes
- Now: Line search automatically choose a valid step size and we have an easy life



Rosenbrock: Gradient Descent with Armoji Linesearch
Gradient Descent with Armoji
final error: 0.01659552824423453
x
y
3.0
2.5
2.0
1.5
1.0
0.5
0.0
−0.5
−1.0
−2.0
−1.5
−1.0
−0.5
0.0
0.5
1.0
1.5
2.0

```julia
begin
    # Rosenbrock function with x* = [a,a^2], f(x*)=0
    a = 1
    b = 100
    h = (x -> (a-x[1])^2 + b*(x[2]-x[1]^2)^2)

    x0 = [-1.,0.5]

    # Gradient Descent with Armijo1
    res_gd_2d_rb_arm1 = gradient_descent_wolfe(h, x0, 1000)
    res_gd_2d_rb_arm1_x = [
        res_gd_2d_rb_arm1[2][i][1] for i=1:length(res_gd_2d_rb_arm1[2])
    ]
    res_gd_2d_rb_arm1_y = [
        res_gd_2d_rb_arm1[2][i][2] for i=1:length(res_gd_2d_rb_arm1[2])
    ]

    clf()
    Δ = 0.1
    X=collect(-2:Δ:2)
    Y=collect(-1:Δ:3)
    F=[h([X[j],Y[i]]) for i=1:length(X), j=1:length(Y)]
    contourf(X,Y,F, levels=50)
    PyPlot.title("Rosenbrock: Gradient Descent with Armoji Linesearch")

    # res_gd_2d_rb
    PyPlot.plot(res_gd_2d_rb_arm1_x, res_gd_2d_rb_arm1_y, color="yellow")
    scatter(res_gd_2d_rb_arm1_x, res_gd_2d_rb_arm1_y, color="yellow")
    # for i=1:length(res_gd_2d_rb_arm1_x)
    #   annotate(string(i), [res_gd_2d_rb_arm1_x[i], res_gd_2d_rb_arm1_y[i]],
    color="w", zorder=2)
    # end

    legend(["Gradient Descent with Armoji"])

    PyPlot.text(2-0.2, -1+0.2, "final error: $(norm(res_gd_2d_rb_arm1[2][end]-
    [1,1]))", size=16,
         ha="right", va="bottom",
        bbox=Dict("boxstyle"=>"square")
    )

    xlabel("x")
    ylabel("y")

    # Mark minimum
    scatter(a, a^2, color="r", s=500, zorder=3, marker="x")

    gcf()
end
```

```
α = 0.001953125
α = 0.001953125
α = 0.00390625
```

```
α = 0.001953125
α = 0.5
α = 0.015625
α = 0.0078125
α = 0.0078125
α = 0.0078125
α = 0.0078125
α = 0.00390625
α = 1.0
α = 0.0009765625
α = 0.0009765625
α = 0.001953125
α = 0.001953125
α = 0.001953125
α = 0.001953125
α = 0.001953125
α = 0.001953125
α = 0.001953125
α = 0.001953125
α = 0.001953125
α = 0.001953125
α = 0.001953125
α = 0.001953125
α = 0.001953125
α = 0.001953125
```

Works but still not the best convergence...

```
([1.0074, 1.01485], [[-1.0, 0.5], [-0.601562, 0.695312], [-0.752011, 0.565064], [-0.737787
```

```
res_gd_2d_rb_arm1 # still not very fast (x*=[1,1]) 😓 (but robust!)
```