penalty

June 16, 2020

1 Penalty and barrier methods

[ ]:
```
using BenchmarkTools
using ForwardDiff
```

[ ]:
```
include("btr.jl")
```

1.1 Penalty methods

1.2 First example

$$\min_x f(x) = x_1^2 + x_1 \sin(x_2) + 4x_2^2$$

[ ]:
```
f(x) = x[1]^2+x[1]*sin(x[2])+4x[2]^2
```

[ ]:
```
using Plots
#plotly()
gr()

default(size=(600,600), fc=:heat)
x, y = -1.0:0.05:1.0, -1.0:0.05:1.0
z = Surface((x,y)->f([x,y]), x, y)
surface(x,y,z, linealpha = 1)
```

[ ]:
```
Plots.contour(x,y,z, linealpha = 0.5, levels=1200)
```

[ ]:
```
g = x -> ForwardDiff.gradient(f, x);
H = x -> ForwardDiff.hessian(f, x)

function g!(storage::Vector, x::Vector)
    storage[1:length(x)] = g(x)
end

function H!(storage::Matrix, x::Vector)
    n = length(x)
    storage[1:n,1:n] = H(x)
end
```

[ ]:
```
state = btr(f, g!, H!, TruncatedCG, [1.0,1.0])
```

[ ]:
```
state.x
```

It is easy to see that $(0, 0)$ is indeed a first-order critical point.

[ ]:
```
g(state.x)
```

Introduce now the constraint

$$(x-1)^2 + (y-1)^2 = 1$$

[ ]:
```
c(x) = (x[1]-1.0)^2+(x[2]-1.0)^2-1.0
gc = x -> ForwardDiff.gradient(c, x);
Hc = x -> ForwardDiff.hessian(c, x)

function gc!(storage::Vector, x::Vector)
    storage[1:length(x)] = gc(x)
end

function Hc!(storage::Matrix, x::Vector)
    n = length(x)
    storage[1:n,1:n] = Hc(x)
end
```

[ ]:
```
  = 1.0
Φ(x) = f(x)+1/(2* )*(c(x)*c(x))
```

[ ]:
```
gΦ = x -> ForwardDiff.gradient(Φ, x);
HΦ = x -> ForwardDiff.hessian(Φ, x)

function gΦ!(storage::Vector, x::Vector)
    storage[1:length(x)] = gΦ(x)
end

function HΦ!(storage::Matrix, x::Vector)
    n = length(x)
    storage[1:n,1:n] = HΦ(x)
end
```

[ ]:
```
state = btr(Φ, gΦ!, HΦ!, TruncatedCG, [0.0,0.0])
```

[ ]:
```
state.x
```

[ ]:
```
f(state.x)
```

[ ]:
```
c(state.x)
```

[ ]:
```
Φ(state.x)
```

[ ]:
```
  = 0.1
state = btr(Φ, gΦ!, HΦ!, TruncatedCG, [0.0,0.0])
```

[ ]:
```
state.x
```

[ ]:
```
f(state.x)
```

[ ]:
```
c(state.x)
```

[ ]:
```
  = 0.01
state = btr(Φ, gΦ!, HΦ!, TruncatedCG, [0.0,0.0])
```

[ ]:
```
c(state.x)
```

[ ]:
```
  = 0.00001
state = btr(Φ, gΦ!, HΦ!, TruncatedCG, [0.0,0.0])
```

[ ]:
```
c(state.x)
```

[ ]:
```
  = 0.00001
state = btr(Φ, gΦ!, HΦ!, TruncatedCG, [0.0,1.0])
```

[ ]:
```
f(state.x)
```

[ ]:
```
state.x
```

[ ]:
```
  = 1
  = 0.1
x0 = [0.0,1.0]
state = btr(Φ, gΦ!, HΦ!, TruncatedCG, x0)
println(state.x)
while (  > 1e-6)
    x0 = state.x
    state = btr(Φ, gΦ!, HΦ!, TruncatedCG, x0)
    println(state.x)
      *=
end
```

[ ]:
```
state.x
```

1.3 Second example

$$\min_x \; -5x_1^2 + x_2^2$$
$$\text{s.t. } x_1 = 1$$

$$\Phi(x,\mu) = -5x_1^2 + x_2^2 + \frac{1}{2\mu}(x_1-1)^2$$

$$\nabla_x \Phi(x,\mu) = \begin{pmatrix} -10x_1 + \frac{1}{\mu}(x_1-1) \\ 2x_2 \end{pmatrix}$$

$$\nabla_x \Phi(x,\mu) = 0$$

iff

$$\begin{cases} -10x_1 + \frac{1}{\mu}(x_1-1) = 0 \\ x_2 = 0 \end{cases}$$

$$(-10\mu + 1)x_1 = 1$$

This is equivalent to

$$x_1 = \frac{1}{1-10\mu}$$

if $10\mu \neq 1$. If $10\mu = 1$, then

$$-10x_1 + 10(x_1 - 1) = -10 \neq 0$$

$$\nabla^2_{xx} \Phi(x,\mu) = \begin{pmatrix} -10 + \frac{1}{\mu} & 0 \\ 0 & 2 \end{pmatrix}$$

If $\mu < 0.1$, then $-10 + 1/\mu > 0$. Then $\nabla^2 \Phi(x,\mu)$ is positive definite. Thus the zero of the gradient is a minimizer.

If $\mu > 0.1$, then $-10 + 1/\mu < 0$. Then $\nabla^2 \Phi(x,\mu)$ is indefinite. Thus the zero of the gradient is a saddle point.

If $\mu = 0.1$, then there is no zero of the gradient.

[ ]:
```
 = 1.0
H = [ -10+1/  0 ; 0 2 ]
eigen(H)
```

[ ]:
```
 = 1.0e-10
H = [ -10+1/  0 ; 0 2 ]
eigen(H)
```

[ ]:
```
cond(H)
```

2 Barrier methods

$$\min_x \; -x + 1$$
$$\text{s.t. } x \leq 1$$

$$L(x, \lambda) = -x + 1 + \lambda(x - 1)$$

KKT

$$\begin{aligned} -1 + \lambda &= 0 \\ x - 1 &\leq 0 \\ \lambda(x - 1) &= 0 \\ \lambda &\geq 0 \end{aligned}$$

We obtain $(x^*, \lambda^*) = (1, 1)$.

[ ]:
```
fb(x) = -x[1]+1
gi(x) = x[1]-1
B(x) = fb(x[1]) - *log(1-x[1])
```

$$0 = \nabla B(x) = \nabla f(x) - \mu \frac{-\nabla g(x)}{-g(x)} = \nabla f(x) - \mu \frac{\nabla g(x)}{g(x)}$$

$$-1 - \mu/(x - 1) = 0$$

$$x - 1 = -\mu$$

$$x = 1 - \mu$$

$$\lambda_\mu = -\frac{\mu}{x(\mu) - 1} = -\frac{\mu}{1 - \mu - 1} = 1$$

[ ]:
```
gB = x -> ForwardDiff.gradient(B, x);
HB = x -> ForwardDiff.hessian(B, x)

function gB!(storage::Vector, x::Vector)
    storage[1:length(x)] = gB(x)
end

function HB!(storage::Matrix, x::Vector)
    n = length(x)
    storage[1:n,1:n] = HB(x)
end
```

[ ]:
```
  = 1.0
gB([0.0])
```

[ ]:
```
state = btr(B, gB!, HB!, TruncatedCG, [0.0])
```

[ ]:
```
state.x
```

[ ]:
```
  = 0.1
btr(B, gB!, HB!, TruncatedCG, [0.0])
```

We have to restrict the step! A simple way is to reduce the trust-region radius if a domain problem is found. We can do it using the exception process in Julia.

[ ]:
```
f(x) = try
   log(x)
catch
   +Inf
end
```

[ ]:
```
f(10)
```

[ ]:
```
f(-10)
```

[ ]:
```
function Bexception(x)
    try
        val = fb(x[1]) -  *log(1-x[1])
        return val
    catch
        return +Inf
    end
end
```

[ ]:
```
state = btr(Bexception, gB!, HB!, TruncatedCG, [0.0])
```

[ ]:
```
state.x
```

[ ]:
```
  = 0.1
x = [0.0]
while (  > 1e-10)
    state = btr(Bexception, gB!, HB!, TruncatedCG, x)
    x = state.x
      = - /gi(x)
    println("x = ", x, ",   = ", )
      = 0.1*
end
```

[ ]:
```
x
```

[ ]:
```

```