BasicTrustRegion-tcg

June 16, 2020

1 Basic Trust Region Algorithm

[59]:
```
using BenchmarkTools
using LinearAlgebra
```

BasicTrustRegion will gather the constants of the method.

[61]:
```
struct BasicTrustRegion{T <: Real}
     1:: T
     2:: T
     1:: T
     2:: T
end

function BTRDefaults()
    return BasicTrustRegion(0.01,0.9,0.5,0.5)
end

# We define a type to store the state at a given iteration.
mutable struct BTRState
    iter::Int
    x::Vector
    xcand::Vector
    g::Vector
    step::Vector
    Δ::Float64
     ::Float64

    tol::Float64

    trace
    keepTrace::Bool

    function BTRState()
        state = new()
        state.tol = 1e-6
        state.keepTrace = false
```

```
        return state
    end
end
```

[62]:
```
function acceptCandidate!(state::BTRState, b::BasicTrustRegion)
    # If the iteration is successful, update the iterate
    if (state.  >= b.1)
        return true
    else
        return false
    end
end
```

[62]:
```
acceptCandidate! (generic function with 1 method)
```

[63]:
```
function updateRadius!(state::BTRState, b::BasicTrustRegion)
    if (state.  >= b.2)
        stepnorm = norm(state.step)
        state.Δ = min(1e20,max(4*stepnorm,state.Δ))
    elseif (state.  >= b.1)
        state.Δ *= b.2
    else
        state.Δ *= b.1
    end
end
```

[63]:
```
updateRadius! (generic function with 1 method)
```

1.1 Basic Trust Region Algorithm

[64]:
```
function btr(f::Function, g!::Function, H!::Function, Step::Function,
    x0::Vector, state:: BTRState = BTRState(), ApproxH::Bool = false, verbose::
 ↪Bool = false)

    b = BTRDefaults()
    state.iter = 0
    state.x = x0
    n=length(x0)

    tol2 = state.tol*state.tol

    state.g = zeros(n)
    # A better initialization procedure should be used with quasi-Newton␣
 ↪approximations
    # We could rely on some preconditioner.
    H = zeros(n,n)+I
```

```
    fx = f(x0)
    g!(x0, state.g)
    state.Δ = 0.1*norm(state.g) # 1.0

    if (ApproxH)
        y = zeros(n)
        gcand = zeros(n)
        # H!(H, y, state.step)
    else
        H!(x0, H)
    end

    nmax = 1000
    if (state.keepTrace)
        state.trace= x0'
    end

    function model(s::Vector, g::Vector, H::Matrix)
        return dot(s, g)+0.5*dot(s, H*s)
    end

    while (dot(state.g,state.g) > tol2 && state.iter < nmax)
        # Compute the step by approximately minimize the model
        state.step = Step(state.g, H, state.Δ)
        state.xcand = state.x+state.step

        # Compute the actual reduction over the predicted reduction
        fcand = f(state.xcand)
        state.  = (fcand-fx)/(model(state.step, state.g, H))

       if (ApproxH)
            g!(state.xcand, gcand)
            y = gcand-state.g
            sy = dot(state.step,y)
#            if (sy < 1e-6)
#                println(state.iter, ". ", state. , " ", state.Δ, " ",␣
↪norm(state.step), " ", (model(state.step, state.g, H)), " ", norm(y), " ",␣
↪sy, " ", norm(state.g))
#            end
            H = H!(H, y, state.step)
        end

        if (acceptCandidate!(state, b))
            state.x = copy(state.xcand)
            if (ApproxH == false)
                g!(state.x, state.g)
                H!(state.x, H)
```

```
            else
                state.g = copy(gcand)
            end
            fx = fcand
        end

        if (state.keepTrace)
            state.trace= [state.trace ; state.x']
        end

        updateRadius!(state, b)
        state.iter += 1
    end

    return state
end
```

[64]: btr (generic function with 4 methods)

[65]:
```
function CauchyStep(g::Vector, H::Matrix, Δ::Float64)
    q = dot(g,H*g)
    normg = norm(g)
    if (q <= 0)
          = 1.0
    else
          = min((normg*normg*normg)/(q*Δ),1.0)
    end
    return - *g*Δ/normg
end
```

[65]: CauchyStep (generic function with 1 method)

[29]:
```
function BFGSUpdate(B::Matrix, y::Vector, s::Vector)
    sy = dot(s,y)
    if (sy > 1e-8)
        Bs = B*s
        B = B - (Bs*Bs')/dot(s, Bs) + (y*y')/sy
    end
    return B
end
```

[29]: BFGSUpdate (generic function with 1 method)

1.2 Truncated Conjugate Gradient

[30]:
```
function stopCG(normg::Float64, normg0::Float64, k::Int, kmax::Int,  ::Float64␣
↪= 0.1,  ::Float64 = 0.5)
    if ((k == kmax) || (normg <= normg0*min( , normg0^ )))
        return true
    else
        return false
    end
end
```

[30]:
```
stopCG (generic function with 3 methods)
```

[66]:
```
function TruncatedCG(g::Vector, H::Matrix, Δ::Float64)
    n = length(g)
    s = zeros(n)

    normg0 = norm(g)
    v = g
    d = -v
    gv = dot(g,v)
    norm2d = gv
    norm2s = 0
    sMd = 0
    k = 0
    Δ2 = Δ*Δ

    while (stopCG(norm(g), normg0, k, n) == false)
        Hd = H*d
          = dot(d,Hd)

        # Is the curvature negative in the direction d?
        if (  <= 0)
            if (k == 0)
                s = d/norm(d)*Δ
            else
              = (-sMd+sqrt(sMd*sMd+norm2d*(Δ2-dot(s,s))))/norm2d
            s +=  *d
            end
            break
        end

          = gv/

        # Check is the model minimizer is outside the trust region
        norm2s +=  *(2*sMd+ *norm2d)
        if (norm2s >= Δ2)
```

```
            if (k == 0)
                s = d/norm(d)*Δ
            else
              = (-sMd+sqrt(sMd*sMd+norm2d*(Δ2-dot(s,s))))/norm2d
            s +=  *d
            end
            break
        end

        # The model minimizer is inside the trust region
        s +=  *d
        g +=  *Hd
        v = g
        newgv = dot(g,v)
          = newgv/gv
        gv = newgv
        d = -v+ *d

        sMd =  *(sMd+ *norm2d)
        norm2d = gv+ * *norm2d

        k += 1;
    end

    return s
end
```

[66]: TruncatedCG (generic function with 1 method)

1.3 Example

[67]:
```
function rosenbrock(x::Vector)
    return (1.0 - x[1])^2 + 100.0 * (x[2] - x[1]^2)^2
end

function rosenbrock_gradient!(x::Vector, storage::Vector)
    storage[1] = -2.0 * (1.0 - x[1]) - 400.0 * (x[2] - x[1]^2) * x[1]
    storage[2] = 200.0 * (x[2] - x[1]^2)
end

function rosenbrock_hessian!(x::Vector, storage::Matrix)
    storage[1, 1] = 2.0 - 400.0 * x[2] + 1200.0 * x[1]^2
    storage[1, 2] = -400.0 * x[1]
    storage[2, 1] = -400.0 * x[1]
    storage[2, 2] = 200.0
end
```

[67]:
```
rosenbrock_hessian! (generic function with 1 method)
```

[68]:
```
defaultState = BTRState()
```

[68]:
```
BTRState(295869840, #undef, #undef, #undef, #undef, 3.404534667e-315,
1.461753134e-315, 1.0e-6, #undef, false)
```

[69]:
```
defaultState.tol
```

[69]:
```
1.0e-6
```

[35]:
```
state = btr(rosenbrock, rosenbrock_gradient!, rosenbrock_hessian!, CauchyStep,␣
↪[0,0])
```

[35]:
```
BTRState(778, [0.9999990671653469, 0.9999981306218827], [0.9999990671653469,
0.9999981306218827], [-3.817981544068388e-7, -7.419362679783603e-7],
[-5.970170605448271e-8, 3.0722752931776427e-8], 0.5578315950889445,
1.0000000474084514, 1.0e-6, #undef, false)
```

[70]:
```
state = btr(rosenbrock, rosenbrock_gradient!, rosenbrock_hessian!, TruncatedCG,␣
↪[0,0])
```

[70]:
```
BTRState(20, [0.9999999999717916, 0.9999999999083069], [0.9999999999717916,
0.9999999999083069], [1.4054073682203108e-8, -7.055245276887945e-9],
[5.939376790494293e-6, 1.1902429996984944e-5], 0.31807381938434265,
1.0000047382691175, 1.0e-6, #undef, false)
```

[71]:
```
defaultState.tol = 1e-4
```

[71]:
```
0.0001
```

[72]:
```
state = btr(rosenbrock, rosenbrock_gradient!, rosenbrock_hessian!, CauchyStep,␣
↪[0,0], defaultState)
```

[72]:
```
BTRState(772, [0.9999574528409513, 0.9999147382962227], [0.9999574528409513,
0.9999147382962227], [-1.7418821383525582e-5, -3.3839188118278685e-5],
[-2.648300225813218e-6, 1.364048575608009e-6], 0.5578315950889445,
1.0000021026865589, 0.0001, #undef, false)
```

[73]:
```
state = btr(rosenbrock, rosenbrock_gradient!, rosenbrock_hessian!, TruncatedCG,␣
↪[0,0], defaultState)
```

[73]:
```
BTRState(19, [0.9999940605950011, 0.99998809747831], [0.9999940605950011,
0.99998809747831], [-2.3800789251098873e-6, -4.7493937449516466e-6],
[-1.8479138998099901e-6, 9.280457471001435e-7], 0.31807381938434265,
1.0000014746706374, 0.0001, #undef, false)
```

[74]:
```
@benchmark btr(rosenbrock, rosenbrock_gradient!, rosenbrock_hessian!,␣
↪CauchyStep, [0,0], defaultState)
```

[74]:
```
BenchmarkTools.Trial:
  memory estimate:  796.47 KiB
  allocs estimate:  20844
  --------------
  minimum time:     2.768 ms (0.00% GC)
  median time:      3.686 ms (0.00% GC)
  mean time:        3.861 ms (1.27% GC)
  maximum time:     13.449 ms (27.32% GC)
  --------------
  samples:          1293
  evals/sample:     1
```

[75]:
```
@benchmark btr(rosenbrock, rosenbrock_gradient!, rosenbrock_hessian!,␣
↪TruncatedCG, [0,0], defaultState)
```

[75]:
```
BenchmarkTools.Trial:
  memory estimate:  34.47 KiB
  allocs estimate:  633
  --------------
  minimum time:     75.699  s (0.00% GC)
  median time:      99.050  s (0.00% GC)
  mean time:        107.139  s (1.99% GC)
  maximum time:     3.850 ms (94.58% GC)
  --------------
  samples:          10000
  evals/sample:     1
```

[76]:
```
state = btr(rosenbrock, rosenbrock_gradient!, BFGSUpdate, TruncatedCG, [0,0],␣
↪defaultState, true)
```

[76]:
```
BTRState(24, [0.9999994477848443, 0.9999986767618864], [0.9999994477848443,
0.9999986767618864], [8.641876423233189e-5, -4.376162143771012e-5],
[0.0001536090193170501, 0.00030842583564268263], 0.2061077379303815,
1.0045406013896907, 0.0001, #undef, false)
```

[43]:
```
@benchmark btr(rosenbrock, rosenbrock_gradient!, BFGSUpdate, TruncatedCG,␣
↪[0,0], defaultState, true)
```

[43]:
```
BenchmarkTools.Trial:
  memory estimate:  71.97 KiB
  allocs estimate:  1211
  --------------
  minimum time:     184.801  s (0.00% GC)
  median time:      399.700  s (0.00% GC)
```

```
  mean time:         479.100  s (1.73% GC)
  maximum time:      7.151 ms (94.42% GC)
  --------------
  samples:           10000
  evals/sample:      1
```

[44]:
```
state = btr(rosenbrock, rosenbrock_gradient!, BFGSUpdate, CauchyStep, [0,0],␣
↪defaultState, true)
```

[44]:
```
BTRState(1000, [0.998549498254876, 0.9970979699362131], [0.998549498254876,
0.9970979699362131], [-0.0016506082846362746, -0.0006261057703182971],
[-1.2580477845551202e-6, 3.3157624308423403e-6], 0.11966392415732387,
0.9999223175223164, 0.0001, #undef, false)
```

"Trust-Region Methods", Introduction

$$f(x, y) = -10x^2 + 10y^2 + 4\sin(xy) - 2x + x^4$$

[77]:
```
using ForwardDiff
defaultState.tol = 1e-6
```

[77]:
```
1.0e-6
```

[78]:
```
f(x::Vector) = -10*x[1]^2+10*x[2]^2+4*sin(x[1]*x[2])-2*x[1]+x[1]^4
```

[78]:
```
f (generic function with 1 method)
```

[47]:
```
g = x -> ForwardDiff.gradient(f, x);
H = x -> ForwardDiff.hessian(f, x)

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

[47]:
```
g! (generic function with 1 method)
```

[79]:
```
function H!(x::Vector, storage::Matrix)
    s = H(x)
    n, m = size(s)
    storage[1:n,1:m] = s[1:length(s)]
end
```

[79]:
```
H! (generic function with 1 method)
```

[80]:
```
state = btr(f, g!, H!, CauchyStep, [0,0])
```

[80]:
```
BTRState(10, [2.306630127652979, -0.3323086483268177], [2.306630127652979,
-0.3323086483268177], [-1.9235244508308824e-9, 1.4039548190680762e-8],
[-2.6188011584984253e-8, -3.5879612974955863e-9], 0.9876566190586658,
1.3810396868170254, 1.0e-6, #undef, false)
```

[81]:
```
state = btr(f, g!, H!, TruncatedCG, [0,0])
```

[81]:
```
BTRState(9, [2.306630127704536, -0.3323086487344874], [2.306630127704536,
-0.3323086487344874], [4.522604513113038e-11, -9.29194499121877e-11],
[2.6931294018039604e-7, -2.159080370597743e-6], 0.9876566190586676,
1.0000768267343858, 1.0e-6, #undef, false)
```

[82]:
```
state = btr(f, g!, BFGSUpdate, CauchyStep, [0,0], BTRState(), true)
```

[82]:
```
BTRState(15, [2.3066301409821888, -0.3323086308230537], [2.3066301409821888,
-0.3323086308230537], [5.99808672063773e-7, 6.325624042347044e-7],
[1.2207055456836203e-6, 3.2358698696715524e-8], 0.13856970761632884,
0.9885297883047421, 1.0e-6, #undef, false)
```

[83]:
```
state = btr(f, g!, BFGSUpdate, TruncatedCG, [0,0], BTRState(), true)
```

[83]:
```
BTRState(16, [2.3066301277215193, -0.33230864871689997], [2.3066301277215193,
-0.33230864871689997], [8.083631541921932e-10, 5.312683626357284e-10],
[3.069381018283615e-8, -9.583908845702118e-7], 0.4040086781927015,
1.000152209177207, 1.0e-6, #undef, false)
```

[84]:
```
f([2.30663, -0.332309])
```

[84]:
```
-31.18073338518543
```

[85]:
```
state = btr(f, g!, H!, TruncatedCG, [-2.0,-2.0], BTRState())
```

[85]:
```
BTRState(9, [-2.2102195200834442, 0.32974845699582234], [-2.2102195200834442,
0.32974845699582234], [-2.2014745582055184e-10, 6.418865439172805e-12],
[2.881632201245532e-6, 8.295428793163734e-8], 0.577574285844766,
0.9999275749910249, 1.0e-6, #undef, false)
```

[86]:
```
f([-2.21022, 0.329748])
```

[86]:
```
-22.14296062774464
```

[87]:
```
st = BTRState()
st.keepTrace = true
```

[87]:
```
true
```

[88]:
```
state = btr(f, g!, BFGSUpdate, TruncatedCG, [-2.0,-2.0], st, true)
```

[88]:
```
BTRState(14, [-2.2102195206812754, 0.32974846415116205], [-2.2102195206812754,
0.32974846415116205], [-1.6023882665194833e-8, 2.356066515218913e-7],
[-6.440813821117127e-9, 7.637090673254499e-8], 0.030643901905232692,
0.9336201299343765, 1.0e-6, [-2.0 -2.0; -2.0 -2.0; … ; -2.2102195142404617
0.32974838778025534; -2.2102195206812754 0.32974846415116205], true)
```

[89]:
```
st.trace
```

[89]:
```
15×2 Array{Float64,2}:
 -2.0      -2.0
 -2.0      -2.0
 -2.1974   -0.183741
 -2.1974   -0.183741
 -2.1974   -0.183741
 -2.43193   0.597025
 -2.08508   0.244409
 -2.19216   0.304055
 -2.20917   0.340282
 -2.21238   0.328408
 -2.20988   0.329765
 -2.21022   0.329742
 -2.21022   0.329749
 -2.21022   0.329748
 -2.21022   0.329748
```

[ ]:

[ ]: