univariate

June 16, 2020

1 Optimization of univariate functions

Consider the following function to optimize.

$$f(x) = x^2 + x - 2\sqrt{x}$$

[1]:
```
f(x) = x^2 + x - 2*sqrt(x)
```

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

I just created a very cool function!

We can rewrite this function as

[2]:
```
g(x::Float64) = x*(x+1) - 2*sqrt(x)
```

[2]:
```
g (generic function with 1 method)
```

[3]:
```
g(x::Int) = 1+1
```

[3]:
```
g (generic function with 2 methods)
```

[4]:
```
g(1.0)
```

[4]:
```
0.0
```

[5]:
```
g(1)
```

[5]:
```
2
```

[6]:
```
methods(g)
```

[6]:
```
# 2 methods for generic function "g":
[1] g(x::Int64) in Main at In[3]:1
[2] g(x::Float64) in Main at In[2]:1
```

[12]:
```
using Plots

pyplot()
```

```
Info: Precompiling PyPlot [d330b81b-6aea-500a-939a-2ce795aea3ee]
@ Base loading.jl:1273
```

[12]:
```
Plots.PyPlotBackend()
```

[13]:
```
xmin = 0.0
xmax = 1.5
plot(g, xmin, xmax)
```

[13]:

1.1 Optimization with the Fibonacci method

Compute the Fibonacci numbers.

[ ]:
```
N = 50
F = ones(N)

for i = 3:N
    F[i] = F[i-1] + F[i-2]
end

F
```

[ ]:
```
F[length(F)]
```

```
[ ]: F[0]
```

Assume that we know that the solution is in [0,1].

```
[ ]: xmin = 0
     xmax = 1.0

     verbose = true
```

```
[ ]: function fibonacci(g::Function, xmin, xmax, verbose::Bool = false)
         k = 1
         i = 1
         d = xmax - xmin
         xG = xmin+(F[N-2]/F[N])*d
         xD = xmin+(F[N-1]/F[N])*d
         fG = g(xG)
         fD = g(xD)

         if (verbose)
             println("Iteration $k.\nxmin = $xmin, xmax = $xmax")
             println("xG = $xG, fG = $fG")
             println("xD = $xD, fD = $fD")
             println("d = $d")
         end

         while (k < N-2)
             k += 1
             i += 1
             if fG < fD
                 xmax = xD
                 d = xmax - xmin
                 xD = xG
                 fD = fG
                 xG = xmin+(F[N-k-1]/F[N-k+1])*d
                 fG = g(xG)
             elseif fG > fD
                 xmin = xG
                 d = xmax - xmin
                 xG = xD
                 fG = fD
                 xD = xmin+(F[N-k]/F[N-k+1])*d
                 fD = g(xD)
             elseif fG == fD
                 k += 1
                 if (k < N-2)
                 xmin = xG
                 xmax = xD
```

```
            d = xmax - xmin
            xG = xmin+(F[N-k-1]/F[N-k+1])*d
            xD = xmin+(F[N-k]/F[N-k+1])*d
            fG = g(xG)
            fD = g(xD)
            end
        end

        if verbose
            println("Iteration $i.\nxmin = $xmin, xmax = $xmax, $k = k")
            println("xG = $xG, fG = $fG")
            println("xD = $xD, fD = $fD")
            println("d = $d")
        end
    end

    return [xmin, xmax]
end
```

[ ]:
```
methods(fibonacci)
```

[ ]:
```
bounds = fibonacci(g, xmin, xmax, true)
```

[ ]:
```
bounds[2]-bounds[1]
```

[ ]:
```
function golden(f::Function, a, b, tol::Float64 = 1e-6)
    k = 1
    i = 1
    if (b < a)
        t = b
        b = a
        a = t
    end
    d = b - a

    # Golden ratio
    gr = (sqrt(5) + 1) / 2

    c = b - d / gr
    d = a + d / gr

    while (abs(c - d) > tol)
        if f(c) < f(d)
            b = d
        else
            a = c
        end
```

```julia
        c = b - (b - a) / gr
        d = a + (b - a) / gr
    end

    return a, b, (b + a) / 2
end
```

[ ]:
```julia
golden(g, 0.0, 1.0)
```

[ ]:
```julia
golden(g, 0.0, 1.0, 1e-8)
```

1.2 Library Optim in Julia

Some optimization routines are directly available in Julia, and can be obtained with the command

[ ]:
```julia
using Pkg
Pkg.add("Optim")

using Optim
```

The basic routine is optimize, taking as first argument the function to minimize, and for univariate functions, second and third arguments the initial lower and upper bound of the search interval.

1.2.1 Golden section

The Golden Section search method is an extension of the Fibonacci approach, where N is not specified, and is taken as $N \to \infty$. We specify it as the fourth argument.

[ ]:
```julia
result = optimize(g, 0, 1, GoldenSection())
```

[ ]:
```julia
Optim.minimizer(result)
```

[ ]:
```julia
bounds
```

1.3 Methods using derivatives

The derivate of f is

$$f'(x) = 2x + 1 - \frac{1}{\sqrt{x}}$$

which can be translated in Julia as

[3]:
```julia
df(x) = 2x+1-1.0/sqrt(x)
```

[3]: df (generic function with 1 method)

Set $f'(x) = 0$, i.e

$$\frac{1}{\sqrt{x}} = 2x + 1$$

or

$$\frac{1}{x} = 4x^2 + 4x + 1$$

We therefore have to look for the roots of polynomial

$$4x^3 + 4x^2 + x - 1 = 0.$$

Not easy! We will use the roots finding library.

[14]:
```
#  Pkg.add("Roots")
using Roots
```

[ ]:
```
h(x) = x*(4x*(x+1)+1)-1
```

The function fzeros aim to find all the roots of a polynomial, but it is quite slow. We will just look for one zero of the function, in the interval $[0, 1]$.

[ ]:
```
?fzero
```

[ ]:
```
fzero(h, 0, 1)
```

[ ]:
```
fzeros(h, 0, 1)
```

1.3.1 Bisection method

We can do it explicitely by coding our bisection function.

[ ]:
```
function bisection(f::Function, a::Float64, b::Float64,  ::Float64 = 1e-8)

    k = 1
    if (a > b)
        c = a
        a = b
        b = c
    end

    fa = f(a)
    fb = f(b)
    if fa == 0
        return k, fa, a, a
    elseif fb == 0
        return k, fb, b, b
    end

    if fa*fb > 0
        println("The function must be of opposite signs at the bounds")
```

```
        return
    end

    d = b-a
    c = a+d/2
    fc = f(c)

    while (d >  )
        if (verbose)
            println("$k. a = $a, b = $b, d = $d, c = $c, fc = $fc")
        end
        k += 1
        if (fc == 0)
            a = b = c
            break
        elseif (fc*fa < 0)
            b = c
            fb = fc
        else
            a = c
            fa = fc
        end
        d = b-a
        c = a+d/2
        fc = f(c)
    end

    return k, fc, a, b
end
```

[ ]:
```
methods(bisection)
```

[ ]:
```
X = bisection(df, 0.0, 1.0)
```

[ ]:
```
X = bisection(df, 0.0, 1.0, 1e-11)
```

[ ]:
```
df(1.0)
```

[ ]:
```
df(2.0)
```

[ ]:
```
X = bisection(df, 1.0, 2.0, 1e-11)
```

[ ]:
```
X
```

1.3.2 Newton method

The second derivate of f is

$$f''(x) = 2 + \frac{1}{2}x^{-\frac{3}{2}}.$$

[2]:
```
function d2f(x::Float64)
    return 2+x^(-3/2)/2
end
```

[2]: d2f (generic function with 1 method)

A basic implementation of the Newton approach follows.

[4]:
```
function Newton(f::Function, df::Function, d2f:: Function, xstart::Float64,  ::
 ↪Float64 = 1e-8, nmax::Int64 = 100)
    k = 1
    x = xstart
    if (verbose)
        fx = f(x)
        println("$k. x = $x, f(x) = $fx")
    end
    dfx = df(x)
    while (abs(dfx) >   && k <= nmax)
        k += 1
        dfx = df(x)
        x = x-dfx/d2f(x)
        if (verbose)
            fx = f(x)
            println("$k. x = $x, f(x) = $fx")
        end
    end
end
```

[4]: Newton (generic function with 3 methods)

[5]:
```
verbose = true
Newton(f, df, d2f, 0.1)
```

```
1. x = 0.1, f(x) = -0.5224555320336759
2. x = 0.2101698321896462, f(x) = -0.6625444777611704
3. x = 0.31601466047275417, f(x) = -0.7084236992061186
4. x = 0.3465158588881345, f(x) = -0.71072285402286
5. x = 0.3478083935817193, f(x) = -0.7107265760534248
6. x = 0.3478103847752347, f(x) = -0.7107265760622221
7. x = 0.347810384779931, f(x) = -0.7107265760622221
```

[6]:
```
verbose = true
Newton(f, df, d2f, 100.0)
```

```
1. x = 100.0, f(x) = 10080.0
```

```
        DomainError with -0.42489377655586225:
    sqrt will only return a complex result if called with a complex argument.␣
 ↪Try sqrt(Complex(x)).



        Stacktrace:

         [1] throw_complex_domainerror(::Symbol, ::Float64) at .\math.jl:31

         [2] sqrt at .\math.jl:493 [inlined]

         [3] f at .\In[1]:1 [inlined]

         [4] Newton(::typeof(f), ::typeof(df), ::typeof(d2f), ::Float64, ::
 ↪Float64, ::Int64) at .\In[4]:14

         [5] Newton(::Function, ::Function, ::Function, ::Float64) at .\In[4]:2

         [6] top-level scope at In[6]:2
```

[10]:
```
x0 = 3.0
x1 = x0-df(x0)/d2f(x0)

verbose = true
Newton(f, df, d2f, 0.5)
```

```
1. x = 0.5, f(x) = -0.6642135623730951
2. x = 0.32842712474619007, f(x) = -0.7098797335674298
3. x = 0.34734380660605063, f(x) = -0.710726092865839
4. x = 0.3478101266311549, f(x) = -0.7107265760620742
5. x = 0.3478103847798521, f(x) = -0.7107265760622219
6. x = 0.34781038477993104, f(x) = -0.710726576062222
```

We see that we converge faster to the optimal solution.

1.4 Numerical differentiation

It is not always easy to explicitely compute the derivative of a function. It is however possible to exploit the derivate definition in order to numerically approximate it. Let f be derivable at x. The derivative is defined as

$$f'(x) = \lim_{\epsilon \to 0} \frac{f(x+\epsilon) - f(x)}{\epsilon}$$

We can therefore approximate the derivative by choosing ϵ small enough and computing

$$f'(x) \approx \frac{f(x+\epsilon) - f(x)}{\epsilon}$$

for instance

[11]:
```
  = 1e-4
dgfd(x) = (g(x+ )-g(x))/
```

[11]:
```
dgfd (generic function with 1 method)
```

Applying the bisection method to that approximation, we obtain

[17]:
```
fzero(dgfd, 0, 1)
```

[17]:
```
0.3477603857669904
```

We cannot choose ϵ arbitrarily small, as illustrated below.

[25]:
```
# fd: Finite difference
dffd(x,  ) = (f(x+ )-f(x))/
x = 1.0
errfd( ) = abs(df(x)-dffd(x,  ))
plot(errfd, 1e-14,1e-12)
```

```
UndefVarError: plot not defined



Stacktrace:

 [1] top-level scope at In[25]:5
```

[26]:
```
  = 1e-40
dgfd(x) = (g(x+ )-g(x))/
fzero(dgfd,0,1)
```

[26]:
```
1.0
```

The method can be refined using the central difference, defined as

$$f'(x) \approx \frac{f(x+\epsilon) - f(x-\epsilon)}{2\epsilon}$$

[27]:
```
dfcd(x,  =1e-6) = (f(x+ )-f(x- ))/(2* )
```

[27]:
```
dfcd (generic function with 2 methods)
```

[28]:
```
x = 1.0
errcd( ) = abs(df(x)-dfcd(x,  ))
plot(errcd, 1e-18,0.1e-14)
```

```
UndefVarError: plot not defined


Stacktrace:

 [1] top-level scope at In[28]:3
```

The central difference provides smaller numerical errors, but at the expense of one additional function evaluation.

The numerical derivates are often expensive to compute, especially in multivariate problem, and we will look for automatic differentiation.

[29]:
```
Newton(f, dfcd, d2f, 1.1)
```

```
1. x = 1.1, f(x) = 0.21238230365969724
2. x = 0.17678773862630892, f(x) = -0.6328810373010464
3. x = 0.2942185128994119, f(x) = -0.7040552143383558
4. x = 0.34392708747069356, f(x) = -0.7106930135038015
5. x = 0.34779235362665795, f(x) = -0.7107265753408357
6. x = 0.3478103843909892, f(x) = -0.7107265760622221
7. x = 0.34781038477877974, f(x) = -0.7107265760622219
```

[30]:
```
dfhd(x,  =1e-4) = (dfcd(x+ )-dfcd(x))/
```

[30]:
```
dfhd (generic function with 2 methods)
```

[31]:
```
Newton(f, dfcd, dfhd, 1.1)
```

```
1. x = 1.1, f(x) = 0.21238230365969724
2. x = 0.17677563355014392, f(x) = -0.6328686318097565
3. x = 0.29424761704081137, f(x) = -0.7040626382540115
4. x = 0.343939312993119, f(x) = -0.7106932248221285
5. x = 0.3477929328575259, f(x) = -0.7107265753864391
6. x = 0.34781038647043694, f(x) = -0.7107265760622221
7. x = 0.34781038479398285, f(x) = -0.7107265760622219
```

[34]:
```
using ForwardDiff
```

[33]:
```
import Pkg
Pkg.add("ForwardDiff")
```

```
  Updating registry at
`C:\Users\slash\.julia\registries\General`
  Updating git-repo
`https://github.com/JuliaRegistries/General.git`
 Resolving package versions…
  Updating
`C:\Users\slash\.julia\environments\v1.2\Project.toml`
  [f6369f11] + ForwardDiff v0.10.8
  Updating
`C:\Users\slash\.julia\environments\v1.2\Manifest.toml`
 [no changes]
```

[35]:
```
g2 = x -> ForwardDiff.derivative(f, x)
```

[35]:
```
#3 (generic function with 1 method)
```

[36]:
```
Newton(f, g2, dfhd, 1.1)
```

```
1. x = 1.1, f(x) = 0.21238230365969724
2. x = 0.17677563346369074, f(x) = -0.6328686317211533
3. x = 0.2942476245134814, f(x) = -0.70406264015959
4. x = 0.3439393183705949, f(x) = -0.7106932249149309
5. x = 0.34779293286432433, f(x) = -0.7107265753864397
6. x = 0.34781038649273033, f(x) = -0.7107265760622221
7. x = 0.34781038477972825, f(x) = -0.7107265760622221
```

[37]:
```
errfd(x) = abs(df(x)-g2(x))
plot(errfd, 1,1.1)
```

```
UndefVarError: plot not defined

Stacktrace:

 [1] top-level scope at In[37]:2
```

[ ]: