36-711 – Homework 3 Solutions

1

(a) By construction, the series of intervals $[a_0, b_0], [a_1, b_1], ...$ contain the root x^* of $f(x)$. Let l_n be the length of the interval after n iterations, then $l_n = (\frac{1}{2})^n(b-a)$. It is easy to see $|x_{n+1} - x_n| = l_{n+1}$ and $|x_n - x_{n-1}| = l_n$, therefore $|x_{n+1} - x_n| \leq k \cdot |x_n - x_{n-1}|$, where for the bisection method, $k = \frac{1}{2}$ and equality holds.
Since x_n is the midpoint of $[a_{n-1}, b_{n-1}]$, and x^* is inside of $[a_{n-1}, b_{n-1}]$, we have $|x_n - x^*| \leq (\frac{1}{2})|b_{n-1} - a_{n-1}| = (\frac{1}{2})^n(b-a)$. For any given $\epsilon > 0$, there exists an N such that when $n > N$, the right hand side is less than ϵ. The convergence rate $\alpha = 1$.

(b) IEEE `float` numbers have 8 exponent digits and 24 fraction digits (23bits plus first digit 1). When add/substract two numbers, the smaller one will be first re-scaled to the same magnitude as the larger one. This is done by increasing the exponent, and right shifting the fractions part. If the difference of magnitude is greater than 2^{24}, the smaller number is actually considered to be zero (right shifted more than 24 digits). Assume that x_n and $[a_n, b_n]$ are at the same magnitude scale (they will be after couple of initial iterations), and we consider the single precision floating point case. In `bisection.c`, within the loop there's a statement "`x = x1 - (x1-x0)/2.0;`". While from the result of part(a) we know $|x_n - x_{n-1}| = (\frac{1}{2})^n(b-a)$, therefore within 24 iterations or less, x_n is going to repeat the same x_1. For similar reasons, it take 53 iterations for double floating point numbers.

(c) Draw a straight line from $(x_0, f(x_0))$ to $(x_1, f(x_1))$ and let x be where the line intersects the x-axis, then $x = x_1 - \frac{f(x_1)}{f(x_1)-f(x_0)}(x_1 - x_0)$. Use this to replace $x = (x_0+x_1)/2$ in `bisection.c`. The convergence rate of secant algorithm is between linear and quadratic ($\alpha = 1.618$, proof is too long, skipped).

2

(a)

$$\log p(y_i|\theta) = -\log(\pi s) - \log(1 + (\frac{y_i - \theta}{s})^2)$$

$$\ell'(\theta) = \sum_{i=1}^{n} \frac{d}{d\theta} \log p(y_i|\theta) = \sum_{i=1}^{n} \frac{2(y_i - \theta)}{s^2 + (y_i - \theta)^2}$$

To use Newton-Raphson method to find the root of $\ell'(\theta) = 0$, we need to calculate the second derivative $\ell''(\theta) = \sum_{i=1}^{n} \frac{2[(y_i-\theta)^2 - s^2]}{[s^2+(y_i-\theta)^2]^2}$. (see attached page for the source code)

(b) When $s = 1$, the algorithm is very sensitive to the choice of initial point. It diverges at most initial points, except at around 1 and 2. (τ = 1.0E-6)

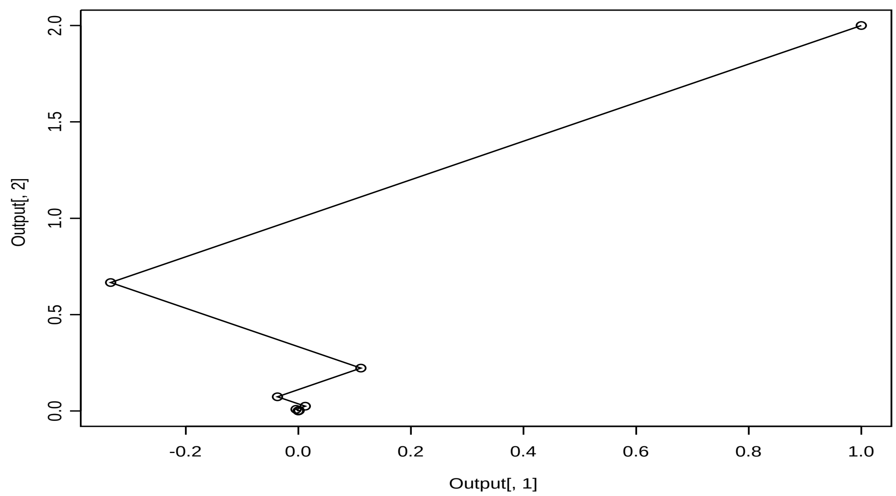
x^*	x_0	iterations	$f'(x^*)$	$f''(x^*)$	$f(x^*)$
1.303075	1	4	0.000000	-10.909700	-16.257844
1.303075	2	6	0.000000	-10.909700	-16.257844

When $s = 0.01$, the algorithm converges in $(0, 5)$ range, diverges at other places. But it also picks some local maximum and minimum points. (τ = 1.0E-10)

x^*	x_0	iterations	$f'(x^*)$	$f''(x^*)$	$f(x^*)$
0.000656	0	4	0.000000	-19732.913029	-55.904320
1.001060	1	4	0.000000	-19175.097591	-37.988150
1.202064	1.2	4	0.000000	-17090.760189	-32.950220
<u>1.299854</u>	1.3	3	-0.000000	-39545.082870	-27.812180
1.397125	1.4	5	-0.000000	-15157.677883	-34.124717
2.271505	2	6	0.000000	139.464530	-56.230412
4.278341	3	9	0.000000	21.530827	-74.811102
4.278341	4	7	0.000000	21.530827	-74.811102

3

(a) The derivative in the direction of $\underline{q}$ is $\frac{df(\underline{x}+t\cdot\underline{q})}{dt}|_{t=0} = \sum_{i=1}^{m} \frac{\partial f(\underline{x}+t\cdot\underline{q})}{\partial x_i} \frac{d(\underline{x}^{(i)}+t\cdot\underline{q}^{(i)})}{dt}|_{t=0} = \nabla f(\underline{x}) \cdot \underline{q} = \underline{q}^T \nabla f(\underline{x})$. When $\underline{q} = \nabla f(\underline{x})$, the inner product reaches the maximum. Therefore $\nabla f(\underline{x})$ is the direction of maximum increase at $\underline{x}$.



(b)
```
f_function(x) {
    2*x[1]*x[1]+x[2]*x[2]
}

df_function(x) {
    c(4*x[1], 2*x[2])
}

ff_function(u) {
    f(X0 + u*df(X0))
}

X0_c(1,2)
Xnew_c(1,2)
Output_NULL

while(1){
    X0_Xnew
    Output_rbind(Output, X0)
```

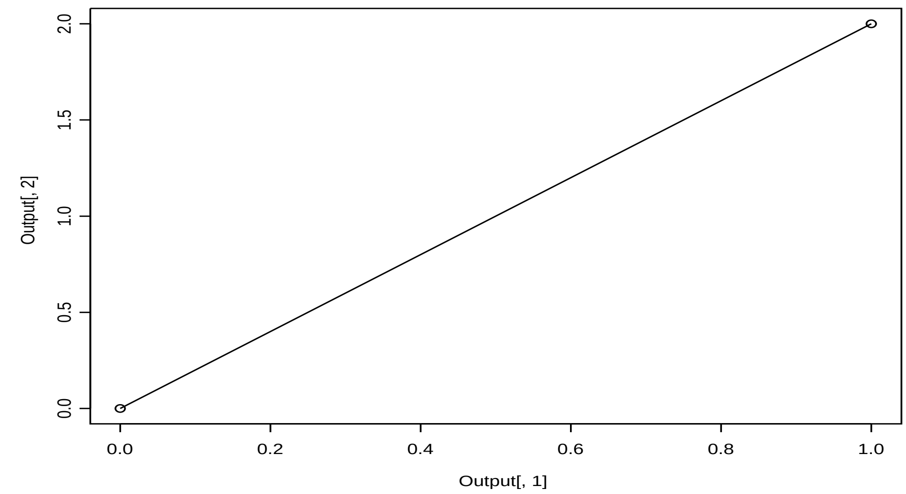
```
    u_nlmin(ff, 0)$x
    Xnew_X0+u*df(X0)
    if (sqrt((Xnew-X0)%*%(Xnew-X0))<1e-6 || f(X0)<1e-6) break
}

plot(Output[,1], Output[,2])
lines(Output[,1], Output[,2])
```

(c) Given $f^*(u) = f(\mathbf{x}+u\mathbf{d})$, as shown in part(a), the three numbers we need can be evaluated as $f^*(0) = f(\mathbf{x})$, $f^*(1) = f(\mathbf{x}+\mathbf{d})$, and $f^{*'}(0) = \mathbf{d}^T\nabla f(\mathbf{x})$.
Suppose $f^*(u)$ is approximated by a quadratic form au^2+bu+c that fits the above values. Then $f^*(u)$ is minimized at $u = -\frac{b}{2a}$. Since $\{f^*(0) = c, f^*(1) = a+b+c, f^{*'}(0) = b\}$, we have $a = f^*(1) - f^*(0) - f^{*'}(0)$ and $b = f^{*'}(0)$.
Therefore the optimal $u_{min} = -\frac{1}{2}\frac{f^{*'}(0)}{f^*(1)-f^*(0)-f^{*'}(0)}$.

(d) Replace the statements `u_nlmin(ff, 0)$x` and `Xnew_X0+u*df(X0)` in part(b) with

```
d_-c(1/4, 1/2)*df(X0)                           # H^(-1) %*% Gradient f
u_-1/2*(d%*%df(X0))/(f(X0+d)-f(X0)-(d%*%df(X0)))  # the u in part(c)
Xnew_X0+u*d                                     # as in Monahan, 8.5.4
```

Then the program converges to $(0,0)$ at one step, no zigzag. Given $f(\mathbf{x})$ is quadratic, the Hessian matrix is constant for all $\mathbf{x}$, and positive definite in this case, and $\nabla f(\mathbf{x}_{new}) = \nabla f(\mathbf{x}_{old} - \mathbf{H}^{-1}(\mathbf{x}_{old})\nabla f(\mathbf{x}_{old})) = \nabla f(\mathbf{x})\cdot(\mathbf{1}-\mathbf{H}^{-1}(\mathbf{x}_{old})\mathbf{H}(\mathbf{x}))|_{\mathbf{x}=\mathbf{x}_{old}} \equiv 0$, therefore after only one iteration, $\mathbf{x}_{new}$ is going to be the local minimum. This also indicates the optimal $\mathbf{u}$ in the above program ought to be 1.



Q2 (b) C program

```
/*****************************************************************
  gcc -lm -o root-nr root-nr.c
  Newton-Raphson for a single variable
  *****************************************************************/

#include<stdlib.h>
#include<stdio.h>
#include<math.h>

typedef double real;

real y[10]={1.2, 2.4, 1.3, 1.3, 0.0, 1.0, 1.8, 0.8, 4.6, 1.4};

real f(real x, real s) {
  int i;
  real sum = 0;

  for(i=0; i<10; i++)
    sum += -log(3.1415926*s)-log(1 + (y[i]-x)*(y[i]-x)/(s*s) );

  return sum;
}

real df(real x, real s) {
  int i;
  real sum = 0;

  for(i=0; i<10; i++)
    sum += 2*(y[i]-x) / ( s*s + (y[i]-x)*(y[i]-x) );

  return sum;
}

real ddf(real x, real s) {
  int i;
  real sum = 0;

  for(i=0; i<10; i++)
    sum += 2*((y[i]-x)*(y[i]-x) - s*s) /
           (s*s+(y[i]-x)*(y[i]-x)) / (s*s+(y[i]-x)*(y[i]-x));

  return sum;
}

real nr(real x0, real s, real tol, int maxit) {
  /* apply newton-raphson to find a root of f(x)=0, using the
     stopping criterion that |xnew - xold|/|xold|<tol */

  int i;
  real x;

  i = 0;
  x = x0;

  //printf("%3d: %f\n",i,x0); /* debug */

  do {
    x0 = x;
    x = x0 - df(x0, s)/ddf(x0, s) ;
    printf("%3d: x=%f\t dfx=%f\t ddfx=%f\t fx=%f\n",i,x,df(x0,s),ddf(x0,s),f(x0,s));
  } while ((++i<=maxit)&&(fabs(x-x0)>fabs(tol*x0)));

  if(i>maxit) {
    printf("Failed to find root in %d iterations\n",maxit);
    exit(1);
  }

  printf("Number of iterations: %d\n",i);
  return(x);
```

```
}

/*****************************************************************/

main() {
  double s;

  s = 0.01;
  printf("s=0.01\n");
  printf("%f\n", nr(0, s, 1.0e-10, 1000));
  printf("%f\n", nr(1, s, 1.0e-10, 1000));
  printf("%f\n", nr(1.2, s, 1.0e-10, 1000));
  printf("%f\n", nr(1.3, s, 1.0e-10, 1000));
  printf("%f\n", nr(1.4, s, 1.0e-10, 1000));
  printf("%f\n", nr(2, s, 1.0e-10, 1000));
  printf("%f\n", nr(3, s, 1.0e-10, 1000));
  printf("%f\n", nr(4, s, 1.0e-10, 1000));

  s = 1;
  printf("\n\n\ns=1\n");
  printf("%f\n", nr(1, s, 1.0e-10, 1000));
  printf("%f\n", nr(2, s, 1.0e-10, 1000));

  return 0;
}
```