Newton’s Method made Difficult with Open Source

1. Write a computer program that accepts a function and an initial guess as input and then uses Newton’s iteration

\[ x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)} \]

to find a solution to the equation \( f(x) = 0 \).

The main program is written in C, compiled using the GNU C compiler on Linux and performs the following steps when run:

- Prompts for the function \( f \).
- Calls Maxima to find \( f' \).
- Outputs Fortran code to compute \( f, f' \) and \( f/f' \).
- Calls GNU Fortran to compile the code.
- Dynamically loads the compiled code.
- Prompts for the initial guess \( x_0 \).
- Performs Newton’s iterations.

This program illustrates the use of symbolic algebra packages to generate reliable optimized code and the dynamic linking of new computational components into a running program. These are useful techniques. The program `newtma.c` is given as

```c
#include <stdio.h>
#include <stdlib.h>
#include <dlfcn.h>
#include <sys/wait.h>
#include <unistd.h>
#include <string.h>
#include <math.h>
#include "enter.h"

static FILE *maxin,*maxout;
static int pipein[2];
static int pipeout[2];

static char *fortran(char *p){
    static char buffer[32767];
    int i;
    fprintf(maxin,
            "genfort(optimize([%s]))$ print("done")$\n",p);
    fflush(maxin);
    i=0;
    buffer[i]=0;
    while(enter(&buffer[i],sizeof(buffer)-i,maxout,1)){
        if(buffer[i+1]==\')\') i+=strlen(&buffer[i]);
        else if(buffer[i+1]==\'\'\') i+=strlen(&buffer[i]);
```
else if(buffer[i]=='d') {
    buffer[i]=0;
    return buffer;
}
}

return buffer;

class

static void maxima_init(){
    pid_t cpid;
    if (pipe(pipein) == -1) {
        perror("pipein");
        exit(1);
    }
    if (pipe(pipeout) == -1) {
        perror("pipeout");
        exit(2);
    }
    if ((cpid = fork()) == -1) {
        perror("fork");
        exit(3);
    } else if (cpid==0) {
        close(pipein[1]);
        close(pipeout[0]);
        close(0);
        if(dup2(pipein[0],0) == -1) {
            perror("pipein");
            exit(4);
        }
        close(1);
        if(dup2(pipeout[1],1) == -1) {
            perror("pipeout");
            exit(5);
        }
        execlp("maxima","maxima","--very-quiet",0);
        perror("maxima");
        exit(6);
    }
    close(pipein[0]);
    close(pipeout[1]);
    if(!maxin=fopen(pipein[1],"w")){
        perror("maxin");
        exit(7);
    }
    if(!maxout=fopen(pipeout[0],"r")){
        perror("maxout");
        exit(8);
    }
}
Newton's Method made Difficult with Open Source

```c
  perror("maxout");
  exit(8);
}

  fprintf(maxin,
"genfort(a):=block (\n"
"  for b in a do\n"
"    if atom(b) then 0\n"
"    else if op(b)=",:,\" then\n"
"      fortran(apply(\"=\",args(b)))\n"
"    else if op(b)=\"[\" then genfort(b)\n"
"    else fortran(b))\n"
"  optimprefix:t\n"
);
}

static double newton(
  double x,double (*fodf_)(double *)){
  int n;
  printf("\n%3s %22s %22s\n","n","x_n","x_n-x_n+1");
  for(n=0;n<256;n++){
    double dx=fodf_(&x);
    printf("%3d %22.15g %22.15g\n",n,x,dx);
    x-=dx;
    if(fabs(dx)<=1.0e-14*abs(x)) return x;
  }
  printf("\n...didn't converge...\n");
  return x;
}

static compile(char *buf){
  char *p;
  char code[1024];
  FILE *codefp;
  maxima_init();
  if(!(codefp=fopen("F.f","w"))){
    perror("codefp");
    exit(9);
  }
  sprintf(code,"f=horner(%s,x),\n"
"  Code for f and df Generated by Maxima\n"
"  C\n"
"  f = %s\n"
"  C\n",buf);
  fprintf(codefp,
"    function f(x)\n"
"  \n");
```
Newton's Method made Difficult with Open Source

```c
" implicit real*8 (a-z)\n"
" character l1*%d\n"
" common /fname/l1\n"
" data l1/'%s\0'/\n"
"%s
" end\n\n",strlen(buf)+1,buf,fortran(code));

sprintf(code,"df=horner(diff(%s,x),x)",buf);

fprintf(codefp,
" function df(x)\n"
" implicit real*8 (a-z)\%s"
" end\n\",fortran(code));

sprintf(code,"fodf=horner((%s)/diff(%s,x),x)",buf);

fprintf(codefp,
" function fodf(x)\n"
" implicit real*8 (a-z)\%s"
" end\",fortran(code));

fclose(maxout);
fclose(maxin);
fclose(codefp);
wait(0);

if(system("gfortran -O2 
"-o F.so -fPIC -shared -fbackslash F.f") == -1){
 perror("gfortran");
 exit(10);
}

main(){
 void *dllib;
 char buf[1024],fsopath[1024];
 double (*f_)(double *), (*df_)(double *), (*fodf_)(double *
 double x,xn;
 char *fname_;
 int r;

 printf("Newton -- Solve f(x)=0 Using Newton’s Method\n"
 "Written May 10, 2010 by Eric Olson\n");

 printf("\nEnter the function f(x):\n");
 fgets(buf,sizeof(buf),stdin);
 if((r=strlen(buf)) > 1) {
 buf[r-1]=0;
 printf("\nCompiling and linking f(x) and df(x)...\n\n");
 compile(buf);
 }
```
The program `newtma.c` depends on a version of `fgets` with a timeout that is used to read the output of Maxima. This is implemented in `enter.c` as
Newton’s Method made Difficult with Open Source

```c
12 tcflush(fileno(stdin),TCIFLUSH);
13 siglongjmp(onalarm,1);
14 }
15
16 char *enter(p,n,fp,t) char *p; int n,t; FILE *fp; {
17     char *r;
18     if(sigsetjmp(onalarm,1)) return 0;
19     signal(SIGALRM,&handler);
20     alarm(t);
21     r=fgets(p,n,fp);
22     alarm(0);
23     signal(SIGALRM,SIG_DFL);
24     return r;
25 }
```

where the header file `enter.h` is given by

```c
1 char *enter(char *,int,FILE *,int);
```

The program `newtma.c` can be compiled and linked with the command

```
gcc -O2 -o newtma newtma.c enter.c -ldl
```

The output when run with the command

```
echo -e "cos(x^2)\n2" | ./newtma
```

is

Newton -- Solve f(x)=0 Using Newton’s Method
Written May 10, 2010 by Eric Olson

Enter the function f(x):

Compiling and linking f(x) and df(x)...

cos(x^2)

Enter the starting value x_0:

<table>
<thead>
<tr>
<th>n</th>
<th>x_n</th>
<th>x_n-x_{n+1}</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>2</td>
<td>-0.215922788612664</td>
</tr>
<tr>
<td>1</td>
<td>2.21592278861265</td>
<td>0.0452521393809502</td>
</tr>
<tr>
<td>2</td>
<td>2.1706706492317</td>
<td>-0.000133118539480844</td>
</tr>
<tr>
<td>3</td>
<td>2.17080376777119</td>
<td>4.09638214845787e-09</td>
</tr>
<tr>
<td>4</td>
<td>2.1708037636748</td>
<td>-2.46883064860499e-16</td>
</tr>
</tbody>
</table>

x = 2.1708037636748
f = 7.04487466626769e-16

It should be noted that `newtma.c` generates a file called `F.f` which contains the optimized Fortran code for computing $f$ and $f'$ and well as $f/f'$. In the above case this file is

```c
1 C    Code for f and df Generated by Maxima
```
Newton’s Method made Difficult with Open Source

```c
2 C
3 C f = cos(x^2)
4 C
5 function f(x)
6 implicit real*8 (a-z)
7 character l1*9
8 common /fname/l1
9 data l1/'cos(x^2)\0'/
10 f = cos(x**2)
11 end
12
13 function df(x)
14 implicit real*8 (a-z)
15 df = -2*x*sin(x**2)
16 end
17
18 function fodf(x)
19 implicit real*8 (a-z)
20 t1 = x**2
21 fodf = -cos(t1)/(x*sin(t1))/2.0E+0
22 end
```

An example of using Maxima to create an N-th order Taylor integrating method without dynamic run-time linking may be found at

http://fractal.math.unr.edu/~ejolson/467-09/taylor.pdf

This report was written May 10, 2010 by Eric Olson.