Brandon Takahashi
October 24, 2016
PA01

1 Newton's Method

Solving $f(x) = 0$ where $f(x) = x^2 - 2$ and $x_0 = 1$.

i) Error Analysis ($e_n = x_n - \sqrt{2}$ for $n = 0, .., 8$)

```
n              x                          error

0   1.0000000000000000000e+00  -4.1421356237309514547e-01
1   1.5000000000000000000e+00  8.5786437626904854525e-02
2   1.4166666666666674068e+00  2.4531042935715952069e-03
3   1.4142156862745098664e+00  2.1239014147411694466e-06
4   1.4142135623746898698e+00  1.5947243525715748546e-12
5   1.4142135623730951454e+00  0.0000000000000000000e+00
6   1.4142135623730949234e+00  -2.2204460492503130808e-16
7   1.4142135623730951454e+00  0.0000000000000000000e+00
8   1.4142135623730949234e+00  -2.2204460492503130808e-16
```

ii) Quadratic Convergence?

In this case quadratic convergence does not exist "perfectly". As seen between steps $n = 1$ and $n = 2$ the number of significant digits does not double.

iii) Rounding Error and Numerical Convergence

The rounding error causes Newton's Method to become unstable for the given problem. While attempting to take a next step after the system shows the method has converged (via $error = 0$) a rounding error occurs such that the solution moves away from the convergence point.

iv) Solving for M_n for $n = 1, ..., 4$

The following list gives the values for M_n solved by $M_n = \frac{|e_{n+1}|}{|e_n|^2}$.

```
M1 = 3.3333333333331038872e-01
M2 = 3.5294117646827655177e-01
M3 = 3.5352238448724582209e-01
```

```
M4 = 0.00000000000000000000e+00
```

All values of M for the given case are less than 1.

2 Secant Method

Solving $f(x) = 0$ where $f(x) = x^2 - 2$ and $x_0 = 1$.

i) Error Analysis ($e_n = x_n - \sqrt{2}$ for $n = 0, .., 8$)

```
n              x                          error

0   0.00000000000000000000e+00  -1.41421356237309514547e+00
1   1.00000000000000000000e+00  -4.14213562373095145475e-01
2   2.00000000000000000000e+00  5.85786437626904854525e-01
3   1.33333333333333348136e+00  -8.08822903976166411l6e-02
4   1.40000000000000013323e+00  -1.42135623730950122479e-02
5   1.41463414634146333881e+00  4.20583968368193339415e-04
6   1.41421143847487007505e+00  -2.12389822507041969857e-06
7   1.41421356205732040578e+00  -3.15774739689800298947e-10
8   1.41421356237309536752e+00  2.22044604925031308085e-16
```

ii) Quadratic Convergence?

Quadratic convergence does not occur using this method. As seen between steps $n = 1$ and $n = 2$ the number of significant digits does not increase and between very few steps does the number of significant digits double.

iii) Solving for M_n for $n = 1, ..., 7$

The following list gives the values for M_n solved by $M_n = \frac{|e_{n+1}|}{|e_n|^{1.618}}$.

```
M1 = 2.43820586365651825744e+00
M2 = 1.92149900902253506496e-01
M3 = 8.31407582617175511253e-01
M4 = 4.09995753025747533549e-01
M5 = 6.16225988349691222723e-01
M6 = 4.76513332062796890476e-01
M7 = 5.22934885792810222327e-01
```

All values of M except for the first are less than 1.

3 Fixed Point Iteration

Solving $f(x) = 0$ where $x_{n+1} = h(x_n)$ and $h(x) = x - \frac{f(x)f'(x)}{f'(x)^2 - f(x)f''(x)}$.

i) Show implications

$f(x) = 0$ implies $h(x) = x$:

$$h(x) = x - \frac{0 * f'(x)}{f'(x)^2 - 0 * f''(x)}$$

$$h(x) = x - \frac{0}{f'(x)^2 - 0}$$

$$h(x) = x$$

$h(x) = x$ implies $f(x) = 0$ or $f'(x) = 0$:
If $h(x) = x$ then $h(x) = x - 0$ in some fassion. Therefor $\frac{f(x)f'(x)}{f'(x)^2 - f(x)f''(x)} = 0$ must be satisfied. To achieve this the numerator of the fraction must $= 0$ while the denominator $\neq 0$. If $f(x) = 0$ or $f'(x) = 0$ this condition is satisfied. Further, if $f(x) = 0$ AND $f'(x) = 0$ there is an issue with division by 0:

$$h(x) = x - \frac{0 * 0}{0^2 - 0 * f''(x)}$$

$$h(x) = x - \frac{0}{0 - 0}$$

$$h(x) = x - \frac{0}{0}$$

$$h(x) = undefined$$

ii) $h'(x)$ and Stability of the Result

$$h'(x) = -\frac{f(x)(-2f(x)f''(x)^2 + f(x)f'(x)f'''(x) + f'(x)^2 f''(x))}{(f'(x)^2 - f(x)f''(x))^2}$$

($h'(x)$ calculated via Wolfram Alpha)

$h'(x) = 0$ when $f(x) = 0$ and $f'(x) \neq 0$:

$$h'(x) = -\frac{0 * (-2 * 0 * f''(x)^2 + 0 * f'(x) * f'''(x) + f'(x)^2 f''(x))}{(f'(x)^2 - 0 * f''(x))^2}$$

$$h'(x) = -\frac{0 * (0 + 0 + f'(x)^2 f''(x))}{(f'(x)^2 - 0)^2}$$

$$h'(x) = -\frac{0}{(f'(x)^2)^2}$$

$$h'(x) = 0$$

$h'(x) = 2$ when $f(x) \neq 0$, $f'(x) = 0$ and $f''(x) \neq 0$:

$$h'(x) = -\frac{f(x)(-2f(x)f''(x)^2 + f(x) * 0 * f'''(x) + 0^2 * f''(x))}{(0^2 - f(x)f''(x))^2}$$

$$h'(x) = -\frac{f(x)(-2f(x)f''(x)^2 + 0 + 0)}{(-f(x)f''(x))^2}$$

$$h'(x) = -(-2) * \frac{f(x)(f(x)f''(x)^2)}{(f(x)f''(x))^2}$$

$$h'(x) = 2 * \frac{f(x)^2 f''(x)^2}{(f(x)f''(x))^2}$$

$$h'(x) = 2$$

The above results show that fixed points of h that result in $f(x) = 0$ are stable because $h'(x) = 0 = f(x)$. Further, the fixed points of h that result in $f'(x) = 0$ are unstable because $h'(x) = 2 \neq f'(x)$.

iii) Fixed Point Implementation and Comparison

Error Analysis

n	x	error
0	1.0000000000000000000e+00	-4.1421356237309514547e-01
1	1.3333333333333325932e+00	-8.0880229039761886156e-02
2	1.4117647058823528106e+00	-2.4488564907423349126e-03
3	1.4142114384748700751e+00	-2.1238982250704196985e-06
4	1.4142135623715001990e+00	-1.5949463971764998859e-12
5	1.4142135623730949234e+00	-2.2204460492503130808e-16
6	1.4142135623730951454e+00	0.0000000000000000000e+00
7	1.4142135623730949234e+00	-2.2204460492503130808e-16
8	1.4142135623730951454e+00	0.0000000000000000000e+00

On the given system the Fixed Point Iteration Method performs on-par with Netwon's Method. Both suffer from instability once the error reaches 0. While Newton's Method reaches an error of 0 one step before the Fixed Point Method, this may be due to rounding errors within the system.

iv) Accelerated Rate of Convergence

4 New Function Exploration

Function:

$$f(x) = 2cos(5x) + 2cos(4x) + 6cos(3x) + 4cos(2x) + 10cos(x) + 3$$

i) Newton's Method

```
For x0 = 1:
error0: -4.71975511965976313178e-02  error10: -5.03062336079107552678e-05
error1: -2.45652170654127033345e-02  error11: -2.51543958549316215567e-05
error2: -1.25640858595492677097e-02  error12: -1.25775160495500415436e-05
error3: -6.35858704881964165168e-03  error13: -6.28883624242959626827e-06
error4: -3.19929855661094109109e-03  error14: -3.14443579241263648782e-06
error5: -1.60476618954530358963e-03  error15: -1.57222353402985959292e-06
error6: -8.03677253649492939758e-04  error16: -7.86107650752043696230e-07
error7: -4.02164063744159250291e-04  error17: -3.93087906447320278858e-07
error8: -2.01163630315859265352e-04  error18: -1.96606606106541903500e-07
error9: -1.00602244741843094289e-04

For x0 = 2:
error0: -9.43951023931952626356e-02  error10: -1.55567992775784702530e-03
error1: -6.16583720201511198411e-02  error11: -1.03688538185675582781e-03
error2: -4.06287612771687456359e-02  error12: -6.91152946874051821169e-04
error3: -2.68959724726149929097e-02  error13: -4.60722457185536171664e-04
error4: -1.78521649214427036156e-02  error14: -3.07127968239395698902e-04
error5: -1.18682044583460211129e-02  error15: -2.04743092826475958645e-04
error6: -7.89783128837484582618e-03  error16: -1.36491617516831809098e-04
error7: -5.25899818274178088018e-03  error17: -9.09924687300112111643e-05
error8: -3.50327252602733452136e-03  error18: -6.06605758468603539768e-05
error9: -2.33431488368118777998e-03
```

ii) Fixed Point Method

```
For x0 = 1:
error0: -4.7197551196597631 78e-02  error10: 3.14677173207655869192e-11
error1: 1.7768708227239304 8754e-03  error11: 3.14677173207655869192e-11
error2: 3.2157141134359079 6150e-06  error12: 3.14677173207655869192e-11
error3: 9.8343555521296366 3507e-13  error13: 3.14677173207655869192e-11
error4: 1.9668711104259273 2701e-12  error14: 3.14677173207655869192e-11
error5: 3.9335201762469296 2272e-12  error15: 3.14677173207655869192e-11
error6: 7.8670403524938592 4544e-12  error16: 3.14677173207655869192e-11
error7: 1.5733858660382793 4596e-11  error17: 3.14677173207655869192e-11
error8: 3.14677173207655869192e-11   error18: 3.14677173207655869192e-11
error9: 3.14677173207655869192e-11

For x0 = 2:
error0: -9.4395102393195262 6356e-02  error10: -2.5071035798518437 3319e-06
error1: -4.7527949858801044 0585e-03  error11: -5.3122035357233698 9600e-06
error2: -6.7898500430985109 2330e-06  error12: -2.9081381791584703 9330e-06
error3: 4.8097554063986081 0188e-06   error13: -6.3337454867529174 8623e-06
error4: -1.5932145904784533 7409e-05  error14: -3.1087908558546928 3615e-06
error5: -3.1729864558016629 4167e-07  error15: -8.3435748301496914 7916e-06
error6: -4.7593073837504107 3596e-07  error16: -1.7331209072324327 2805e-06
error7: -7.1317747529775488 1467e-07  error17: -2.3677631006968624 6607e-06
error8: -1.0742473977032318 5439e-06  error18: -2.3677631006968624 6607e-06
error9: -1.6357170937730813 8755e-06
```

iii) Rate of Convergence and Rounding Error

The Fixed Point Method converges faster than Newton's Method. Even so, Newton's Method does not suffer from an error instability that the Fixed Point Method seems to. The Fixed Point Method error peaks out at a particular step for the given function and begins to lose significant digits after this step. This anamoly is most likely a result of rounding error within the system where the system performed multiple rounding errors back to back.

Code and closing remarks

All code was worked on with Mat Boggs, Jason Rush, and myself. Some code has been modified from their original versions.

Newton's Method Error

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

double f(double x)
{
    return x * x - 2;
}

double df(double x)
{
    return 2 * x;
}

double g(double x)
{
    return x - f(x) / df(x);
}

double error(double x)
{
    return x - sqrt(2);
}

int main()
{
    double x = 1, err[9];
    int i;

    err[0] = error(x);
    printf("n\t\t\t\tx\t\t\t\t\terror\n\n");
    printf("%d\t%10.20e\t%10.20e\n", 0, x, err[0]);

    for(i = 1; i < 9; i++)
    {
        x = g(x);

        err[i] = error(x);

        printf("%d\t%10.20e\t%10.20e\n", i, x, err[i]);
    }

    return 0;
}
```

Newton's Method Solving for M

```c
#include <stdio.h>
#include <math.h>

double f(double x)
{
    return x * x - 2;
}

double df(double x)
{
    return 2 * x;
}

double g(double x)
{
    return x - f(x) / df(x);
}

double error(double x)
{
    return x - sqrt(2);
}

double Mn(double en, double en_1)
{
    return fabs(en_1) / fabs(en * en);
}

int main()
{
    double x = 1, err[9], M[5];
    int i;

    err[0] = error(x);

    for(i = 1; i < 9; i++)
    {
        x = g(x);

        err[i] = error(x);
    }

    for(i = 1; i < 5; i++)
    {
        M[i - 1] = Mn(err[i], err[i + 1]);

        printf("\tM%d = %10.20e\n", i, M[i - 1]);
    }
```

```
49
50      return 0;
51  }
```

Secant Method Error

```
1   #include <stdio.h>
2   #include <math.h>
3
4   double f(double x)
5   {
6       return (x * x) - 2;
7   }
8
9   double error(double x)
10  {
11      return x - sqrt(2);
12  }
13
14  int main()
15  {
16      int i = 2;
17
18      double q0, q1, x0 = 0, x1 = 1, x, err[9];
19
20      q0 = f(x0);
21      q1 = f(x1);
22
23      err[0] = error(x0);
24      err[1] = error(x1);
25      printf("n\t\t\t\tx\t\t\t\t\t\terror\n\n");
26      printf("%d\t%10.20e\t%10.20e\n", 0, x0, err[0]);
27      printf("%d\t%10.20e\t%10.20e\n", 1, x1, err[1]);
28
29      for(i = 2; i < 9; i++)
30      {
31          // compute xi
32          x = x1 - q1 * (x1 - x0) / (q1 - q0);
33
34          // update values
35          x0 = x1;
36          q0 = q1;
37          x1 = x;
38          q1 = f(x);
39
40          // calculate error
41          err[i] = error(x);
42
43          printf("%d\t%10.20e\t%10.20e\n", i, x, err[i]);
44      }
```

```
    return 0;
}
```

Secant Method Solving for M

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

double f(double x)
{
    return (x * x) - 2;
}

double error(double x)
{
    return x - sqrt(2);
}

double Mn(double err0, double err1)
{
    return fabs(err1) / pow(fabs(err0), 1.618);
}

int main()
{
    int i = 2;

    double q0, q1, x0 = 0, x1 = 1, x, err[9], M[8];

    q0 = f(x0);
    q1 = f(x1);

    err[0] = error(x0);
    err[1] = error(x1);

    for(i = 2; i < 9; i++)
    {
        // compute xi
        x = x1 - q1 * (x1 - x0) / (q1 - q0);

        // update values
        x0 = x1;
        q0 = q1;
        x1 = x;
        q1 = f(x);

        // calculate error
        err[i] = error(x);
    }

```

```
    for(i = 1; i < 8; i++)
    {
        M[i - 1] = Mn(err[i], err[i+1]);
        printf("\tM%d = %10.20e\n", i, M[i - 1]);
    }

    return 0;
}
```

Fixed Point Error

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

double f(double x)
{
    return x * x - 2;
}

double df(double x)
{
    return 2 * x;
}

double d2f(double x)
{
    return 2.0;
}

double h(double x)
{
    return x - ((f(x) * df(x)) / ((df(x) * df(x)) - (f(x) * d2f(x))));
}

double error(double x)
{
    return x - sqrt(2);
}

int main()
{
    double x = 1, err[9];

    int i;

    err[0] = error(x);
    printf("n\t\t\t\tx\t\t\t\t\terror\n\n");
    printf("%d\t%10.20e\t%10.20e\n", 0, x, err[0]);

    for(i = 1; i < 9; i++)
```

```
40      {
41          x = h(x);
42
43          err[i] = error(x);
44
45          printf("%d\t%10.20e\t%10.20e\n", i, x, err[i]);
46      }
47
48      return 0;
49  }
```

Newton's Method for "New Function"

```
1   #include <stdio.h>
2   #include <math.h>
3
4   double f(double x)
5   {
6       return 2 * cos(5 * x) + 2 * cos (4 * x) + 6 * cos(3 * x) +
7               4 * cos(2 * x) + 10 * cos(x) + 3;
8   }
9
10  double df(double x)
11  {
12      return -10 * sin(5 * x) - 8 * sin(4 * x) - 18 * sin(3 * x) -
13              8 * sin(2 * x) - 10 * sin(x);
14  }
15
16  double d2f(double x)
17  {
18      return -50 * cos(5 * x) - 32 * cos(4 * x) - 54 * cos(3 * x) -
19              16 * cos(2 * x) - 10 * cos(x);
20  }
21
22  double g(double x)
23  {
24      return x - f(x) / df(x);
25  }
26
27  double h(double x)
28  {
29      return x - ((f(x) * df(x)) / ((df(x) * df(x)) - (f(x) * d2f(x))));
30  }
31
32  double errorLow(double x)
33  {
34      return x - M_PI / 3;
35  }
36
37  double errorHigh(double x)
```

```
{
    return x - 2 * M_PI / 3;
}

int main()
{
    double x = 1, err[19];
    int i;

    printf("For x0 = 1:\n");

    for(i = 0; i < 19; i++)
    {
        err[i] = errorLow(x);
        x = g(x);
    }

    for(i = 0; i < 9; i++)
    {
        printf("error%d: %10.20e\t error%d: %10.20e\n",
            i, err[i], i + 10, err[i + 10]);
    }
        printf("error%d: %10.20e\n", 9, err[9]);

    printf("\nFor x0 = 2:\n");

    x = 2;

    for(i = 0; i < 19; i++)
    {
        err[i] = errorHigh(x);
        x = g(x);
    }

    for(i = 0; i < 9; i++)
    {
        printf("error%d: %10.20e\t error%d: %10.20e\n",
            i, err[i], i + 10, err[i + 10]);
    }
        printf("error%d: %10.20e\n", 9, err[9]);

    return 0;
}
```

- Some code has been modified to "look good" w/ LaTeX
- Newton's and Fixed Point code for the "new function" are the same EXCEPT g should be replaced with h.