AM117 Homework set #2, Sample solutions

Blurb from the TA:

- If you are proving something, write down *what* and *how*. It doesn't take much time, but it helps you and me understand what you're writing and tells me that you know what you're talking about.
- Please don't use red pen in your answers.
- If you are asked for the apparent order of convergence, *calculate*, don't guess.

1. Problem 9, p.94

We are considering

$$f(x) := \frac{x^3 + 3\,x\,a}{3x^2 + a}.$$

(Compare this to the second problem of the last homework set.)

Verifying that $\sqrt{a}$ is a fixed point is a matter of plugging it in:

$$f(\sqrt{a}) = \frac{a^{3/2} + 3\,a^{1/2}a}{3a + a} = \frac{4a^{3/2}}{4a} = \sqrt{a}.$$

To obtain the order of convergence "the new way", we obtain the first derivative of f:

$$f'(x) = \frac{3x^4 - 6\,a\,x^2 + 3a^2}{9x^4 + 6\,a\,x^2 + a^2}$$

and observe

$$f'(\sqrt{a}) = \frac{3a^2 - 6\,a\,a + 3a^2}{9a^2 + 6\,a\,a + a^2} = 0.$$

By the theorems in Section 2.3, we may exclude linear convergence, and we need to keep looking.

$$f''(x) = \frac{48\,a\,x^3 - 48\,a^2x}{27x^6 + 27a\,x^4 + 9a^2x^2 + a^3}.$$

It's easy to see that $f''(\sqrt{a}) = 0$, so we don't have quadratic convergence, either. Next:

$$f'''(x) = \frac{-432a\,x^4 + 864a^2x^2 - 48a^3}{81x^8 + 108\,a\,x^6 + 54a^2x^4 + 12\,a^3x^2 + a^4}.$$

We find

$$f'''(\sqrt{a}) = \frac{3}{2a},$$

thus by our theorem, we know (in agreement with last time) that our convergence is of third order. Further, we find the asymptotic error constant

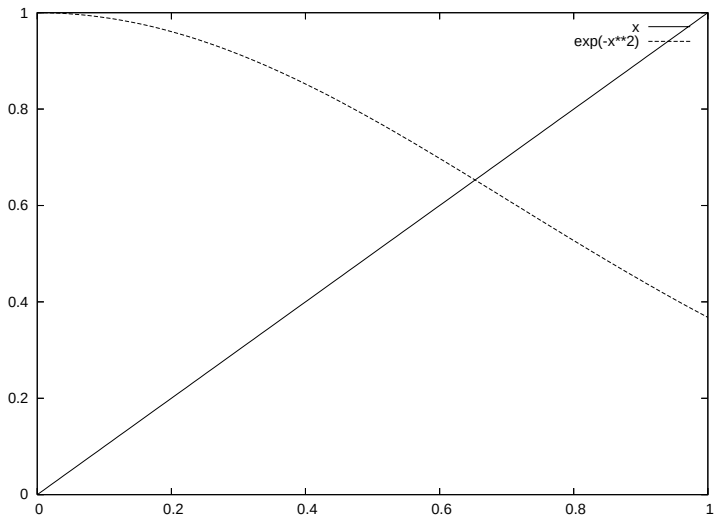
$$\lim_{n\to\infty} \frac{|e_{n+1}|}{|e_n|^3} = \frac{f'''(\sqrt{a})}{3!} = \frac{3/2a}{6} = \frac{1}{4a}$$

agrees as well.

(If you need a computer algebra system to do tedious calculations like these, I can recommend either Maxima[1] or Axiom[2]. Both are open-source and free to download.)

2. Problem 11, p. 95

a) Let's start by visualizing the situation:



We shall apply the theorem on p. 84 to

$$f(x) := \exp(-x^2).$$

We need to verify the following:

- f is continuous on the closed interval $[0,1]$: ✓
- It maps $[0,1]$ to $[0,1]$: ✓ (Note: not necessarily *onto* $[0,1]$.)
- It's differentiable: ✓

  $$f'(x) = -2x\exp(-x^2).$$

- The magnitude of f' is bounded away from 1 on $(0,1)$: Observe $f' \leqslant 0$ on $[0,1]$, $f'(0) = 0$. Consider

  $$f''(x) = \exp(-x^2)(4x^2-2) = 0 \Leftrightarrow x = \pm\sqrt{1/2}.$$

  Let $x_0 := \sqrt{1/2}$ and observe $f'''(x_0) > 0$, so there's a local minimum of f' at x_0. It's not hard to verify

  $$f'(x_0) > -1,$$

  and by $f \in C^\infty(\mathbb{R})$ we know that $f'(x_0) \leqslant f(x)$ for $x \in [0,1]$.

As a consequence, the theorem on p. 84 yields the claim.

b) and c) Let x^* be the fixed point whose existence and uniqueness is guaranteed by a). It's easy to see that $f'(x^*) = 0$, so convergence will be at best linear. We may thus use the error estimates (1) and (2) on p. 92 to estimate our progress.

1. `http://wxmaxima.sourceforge.net/`
2. `http://wiki.axiom-developer.org/FrontPage`

After the first iteration, we use the a-priori bound

$$|p_n - p| \leqslant \frac{k^n}{1-k}|p_1 - p_0|$$

with $k = |f'(x_0)| \approx 0.8578$ to estimate the number n of iterations needed:

$$\frac{\text{tol} \cdot (1-k)}{|p_1 - p_0|} \leqslant k^n \Leftrightarrow \log \frac{\text{tol} \cdot (1-k)}{|p_1 - p_0|} \Big/ \log k \leqslant n$$

We obtain these results, noting that the estimate of the number of iterations is fairly good–it tells us at least the order of magnitude of steps to expect.

```
Estimated number of iterations: 107.276
Iteration:1 value:1
Iteration:2 value:0.367879 error:0.24482
Iteration:3 value:0.873423 error:0.224648
Iteration:4 value:0.466327 error:0.181591
Iteration:5 value:0.804559 error:0.153491
Iteration:6 value:0.523449 error:0.127591
Iteration:7 value:0.760333 error:0.108329
Iteration:8 value:0.56096 error:0.0911151
Iteration:9 value:0.730025 error:0.0775791
Iteration:10 value:0.586879 error:0.0656315
Iteration:11 value:0.708626 error:0.0559563
Iteration:12 value:0.605227 error:0.0474864
Iteration:13 value:0.693295 error:0.0405079
Iteration:14 value:0.618376 error:0.0344371
Iteration:15 value:0.682229 error:0.0293806
Iteration:16 value:0.627861 error:0.0250034
Iteration:17 value:0.674213 error:0.0213315
Iteration:18 value:0.634725 error:0.0181651
Iteration:19 value:0.668395 error:0.0154961
Iteration:20 value:0.639703 error:0.0132011
Iteration:21 value:0.664168 error:0.0112603
Iteration:22 value:0.643316 error:0.00959513
Iteration:23 value:0.661097 error:0.00818367
Iteration:24 value:0.64594 error:0.00697469
Iteration:25 value:0.658864 error:0.00594822
Iteration:26 value:0.647846 error:0.00507008
Iteration:27 value:0.657241 error:0.00432363
Iteration:28 value:0.649232 error:0.00368563
Iteration:29 value:0.656061 error:0.00314285
Iteration:30 value:0.650239 error:0.00267923
Iteration:31 value:0.655203 error:0.00228458
Iteration:32 value:0.650971 error:0.00194765
Iteration:33 value:0.654579 error:0.00166071
Iteration:34 value:0.651503 error:0.00141583
Iteration:35 value:0.654126 error:0.00120721
Iteration:36 value:0.651889 error:0.00102922
Iteration:37 value:0.653796 error:0.000877557
Iteration:38 value:0.65217 error:0.000748182
Iteration:39 value:0.653557 error:0.000637924
Iteration:40 value:0.652375 error:0.000543883
Iteration:41 value:0.653382 error:0.000463728
```

```
Iteration:42 value:0.652523 error:0.000395369
Iteration:43 value:0.653256 error:0.0003371
Iteration:44 value:0.652631 error:0.000287409
Iteration:45 value:0.653164 error:0.000245049
Iteration:46 value:0.65271 error:0.000208928
Iteration:47 value:0.653097 error:0.000178135
Iteration:48 value:0.652767 error:0.000151878
Iteration:49 value:0.653048 error:0.000129493
Iteration:50 value:0.652808 error:0.000110405
Iteration:51 value:0.653013 error:9.41327e-05
Iteration:52 value:0.652838 error:8.02578e-05
Iteration:53 value:0.652987 error:6.84285e-05
Iteration:54 value:0.65286 error:5.83423e-05
Iteration:55 value:0.652968 error:4.97431e-05
Iteration:56 value:0.652876 error:4.24111e-05
Iteration:57 value:0.652955 error:3.616e-05
Iteration:58 value:0.652888 error:3.08302e-05
Iteration:59 value:0.652945 error:2.6286e-05
Iteration:60 value:0.652896 error:2.24116e-05
Iteration:61 value:0.652938 error:1.91083e-05
Iteration:62 value:0.652902 error:1.62918e-05
Iteration:63 value:0.652933 error:1.38905e-05
Iteration:64 value:0.652907 error:1.18431e-05
Iteration:65 value:0.652929 error:1.00975e-05
Iteration:66 value:0.65291 error:8.60918e-06
Iteration:67 value:0.652926 error:7.34023e-06
Iteration:68 value:0.652912 error:6.25832e-06
Iteration:69 value:0.652924 error:5.33588e-06
Iteration:70 value:0.652914 error:4.5494e-06
Iteration:71 value:0.652923 error:3.87884e-06
Iteration:72 value:0.652915 error:3.30712e-06
Iteration:73 value:0.652921 error:2.81967e-06
Iteration:74 value:0.652916 error:2.40407e-06
Iteration:75 value:0.652921 error:2.04972e-06
Iteration:76 value:0.652917 error:1.7476e-06
Iteration:77 value:0.65292 error:1.49002e-06
Iteration:78 value:0.652917 error:1.2704e-06
Iteration:79 value:0.65292 error:1.08315e-06
Iteration:80 value:0.652918 error:9.23497e-07
Iteration:81 value:0.652919 error:7.87379e-07
Iteration:82 value:0.652918 error:6.71323e-07
Iteration:83 value:0.652919 error:5.72374e-07
Iteration:84 value:0.652918 error:4.88009e-07
```

from the following Matlab code:

```
function p2
  tol = 5e-7;
  itcount = 0;
  this = 0;
  last = 0;

  while 1
    last2 = last;
```

```
    last = this;
    this = f(last);
    itcount = itcount + 1;

    if (itcount == 1)
      x0 = sqrt(1/2);
      k = abs(-2*x0*exp(-x0^2));
      est_itcount = log(tol*(1-k)/abs(this-last))/log(k);
      disp(sprintf('Estimated number of iterations: %g', est_itcount))
    end
    if (itcount >= 2)
      fprime = (this-last)/(last-last2);
      err = abs(fprime/(fprime-1))*abs(this-last);
      disp(sprintf('Iteration:%d value:%g error:%g', itcount, this, err))
    else
      disp(sprintf('Iteration:%d value:%g', itcount, this))
    end
    if (itcount > 2) && (err < tol)
      break
    end
  end
end
function y = f(x)
  y = exp(-x^2);
end
```

3. Problem 9, p. 104

We obtain

```
Iteration:0 value:4 error:8.5840734641e-01
Iteration:1 value:2.842178718 error:2.9941393594e-01
Iteration:2 value:3.15087294 error:9.2802860956e-03
Iteration:3 value:3.141592387 error:2.6642673445e-07
Apparent order of convergence at iteration 1:3.88071
Apparent order of convergence at iteration 2:3.23475
For reference: pi=3.141592654
```

On double precision systems, the error is below machine precision from the fourth iteration onwards.

The apparent order of convergence approaches three, which makes sense since $\sin'' x = -\sin x$, which is zero at π, accelerating Newton's method to third order according to the discussion on p. 100.

Here's the code:

```
function p3
  newton(4, @f, @fprime, 3, pi);
  disp(sprintf('For reference: pi=%.10g', pi))
end

function y = f(x)
  y = sin(x);
end
function y = fprime(x)
  y = cos(x);
end
```

```
function newton(x, f, fprime, itcount, true)
  disp(sprintf('Iteration:%d value:%.10g error:%.10e', 0, x, abs(x-true)))
  for i=1:itcount
    x = x - f(x)/fprime(x);
    disp(sprintf('Iteration:%d value:%.10g error:%.10e', i, x, abs(x-true)))
    errors(i) = abs(x-true);
  end

  order_of_convergence(errors)
end

function order_of_convergence(seq)
  for at=1:length(seq)-1
    order = log(seq(at+1)) / log(seq(at+0));
    disp(sprintf('Apparent order of convergence at iteration %d:%g', at, order))
  end
end
```

4. Problem 11, p. 105

Results:

```
Iteration:0 value:0 error:3.3333333333e-01
Iteration:1 value:0.1129032258 error:2.2043010753e-01
Iteration:2 value:0.1871468695 error:1.4618646388e-01
Iteration:3 value:0.2362083272 error:9.7125006136e-02
Iteration:4 value:0.2687288269 error:6.4604506450e-02
Iteration:5 value:0.2903276525 error:4.3005680801e-02
Iteration:6 value:0.3046911229 error:2.8642210472e-02
Iteration:7 value:0.3142510214 error:1.9082311969e-02
Iteration:8 value:0.3206173284 error:1.2716004913e-02
Iteration:9 value:0.3248584524 error:8.4748809797e-03
Iteration:10 value:0.3276845026 error:5.6488307373e-03
Apparent order of convergence at iteration 1:1.27159
Apparent order of convergence at iteration 2:1.21264
Apparent order of convergence at iteration 3:1.17485
Apparent order of convergence at iteration 4:1.14855
Apparent order of convergence at iteration 5:1.12918
Apparent order of convergence at iteration 6:1.11431
Apparent order of convergence at iteration 7:1.10253
Apparent order of convergence at iteration 8:1.09296
Apparent order of convergence at iteration 9:1.08503
```

Discussion: The apparent order of convergence is roughly one, which is the same as the bisection method. Unfortunately, though, the bisection method would converge slightly faster–it would cut its error in half in each iteration, which our method fails to achieve by a slight margin–its convergence in this case is of order $O((1-1/3)^n) = O((2/3)^n)$ (cf. p. 101).

The polyomial can be factored as

$$27\,x\hat{}4 + 162\,x\hat{}3 - 180\,x\hat{}2 + 62\,x - 7 = (27x + 189)(x - 1/3)^3,$$

so 1/3 is a root of multiplicity three.

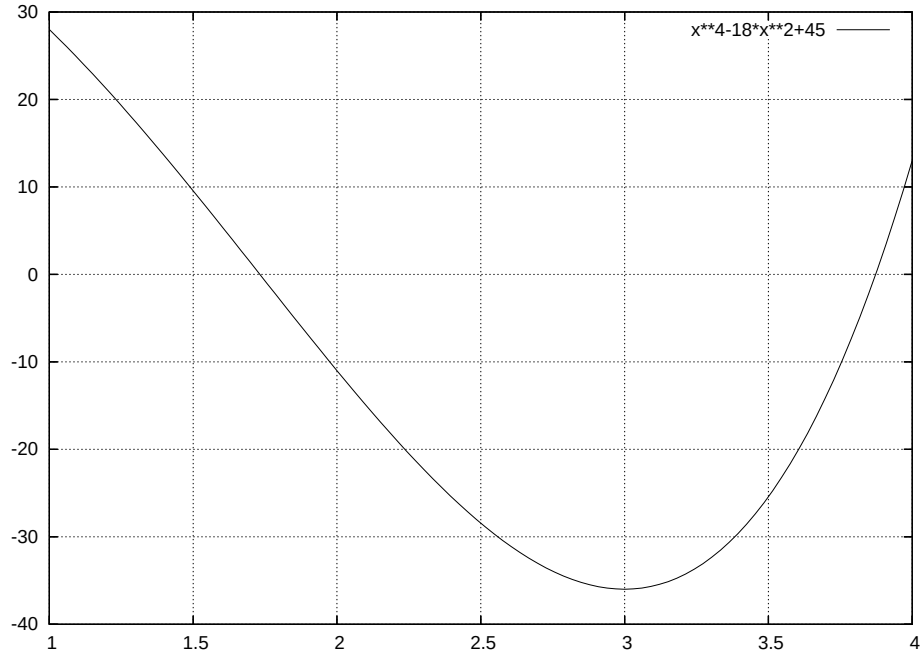
Code:

```
function p4
  newton(0, @f, @fprime, 10, 1./3);
end
function y = f(x)
  y = 27*x^4+162*x^3-180*x^2+62*x-7;
end
function y = fprime(x)
  y = 108*x^3+486*x^2-360*x+62;
end

function newton(x, f, fprime, itcount, true)
  disp(sprintf('Iteration:%d value:%.10g error:%.10e', 0, x, abs(x-true)))
  for i=1:itcount
    x = x - f(x)/fprime(x);
    disp(sprintf('Iteration:%d value:%.10g error:%.10e', i, x, abs(x-true)))
    errors(i) = abs(x-true);
  end

  order_of_convergence(errors)
end

function order_of_convergence(seq)
  for at=1:length(seq)-1
    order = log(seq(at+1)) / log(seq(at+0));
    disp(sprintf('Apparent order of convergence at iteration %d:%g', at, order))
  end
end
```

5. Problem 10, p.112

We begin by plotting the function

$$f(x) := x^4 - 18x^2 + 45.$$



This plot provides sufficient verification of the existence of the roots in parts a) and b), by the Intermediate Value Theorem.

Let's take a look at the data:

```
Part a)
Iteration:1 value:1.717948718 error:1.4102089620e-02
Iteration:2 value:1.732218859 error:1.6805176076e-04
Iteration:3 value:1.732050802 error:5.4926438953e-09
Iteration:4 value:1.732050808 error:0.0000000000e+00
Iteration:5 value:1.732050808 error:2.2204460493e-16
Apparent order of convergence at iteration 1:2.03951
Apparent order of convergence at iteration 2:2.18839
Apparent order of convergence at iteration 3:Inf
Apparent order of convergence at iteration 4:0
Part b)
Iteration:1 value:3.734693878 error:1.3828946866e-01
Iteration:2 value:3.859328134 error:1.3655212414e-02
Iteration:3 value:3.874581141 error:1.5977950841e-03
Iteration:4 value:3.872966331 error:1.7014963079e-05
Iteration:5 value:3.872983325 error:2.1039905373e-08
Iteration:6 value:3.872983346 error:2.7711166695e-13
Iteration:7 value:3.872983346 error:0.0000000000e+00
Apparent order of convergence at iteration 1:2.17025
Apparent order of convergence at iteration 2:1.49969
Apparent order of convergence at iteration 3:1.70542
Apparent order of convergence at iteration 4:1.60971
Apparent order of convergence at iteration 5:1.63572
Apparent order of convergence at iteration 6:Inf
```

Discussion: a) appears to converge of order greater than two, while b) appears to converge at just about the order predicted for the secant method, namely $(1+\sqrt{5})/2 \approx 1.618$. The explanation for the difference in behaviors (or, rather, for the exceptionally good behavior in a)) lies in the fact that f is nearly linear around $\sqrt{3}$, in fact $f''(\sqrt{3}) = 0$. If you look up the error term of the secant method, you will find that it is dominated by the second derivative (cf. p. 109). Also, if you consider that the secant method finds the root of a linear function in a single step, you will agree that this behavior makes sense.

Here's the code:

```
function p5
  disp('Part a)')
  secant(1, 2, @f, 5, sqrt(3))
  disp('Part b)')
  secant(3, 4, @f, 7, sqrt(15))
end
function y = f(x)
  y = x^4-18*x^2+45;
end
function secant(x0, x1, f, itcount, true)
  this = x1;
  last = x0;
  for i=1:itcount
    new = this - f(this)*(this-last)/(f(this)-f(last));
    last = this;
    this = new;
```

```
    disp(sprintf('Iteration:%d value:%.10g error:%.10e', i, this, abs(this-true)))
    errors(i) = abs(this-true);
  end

  order_of_convergence(errors)
end

function order_of_convergence(seq)
  for at=1:length(seq)-1
    order = log(seq(at+1)) / log(seq(at+0));
    disp(sprintf('Apparent order of convergence at iteration %d:%g', at, order))
  end
end
```

6. Problem 12, p.124

Data:

```
----------------------------------------
Known multiplicity:
----------------------------------------
Iteration:0 value:1 error:1.0000000000e+00
Iteration:1 value:-0.05988801046 error:5.9888010461e-02
Iteration:2 value:1.193602343e-05 error:1.1936023426e-05
Iteration:3 value:1.694065895e-21 error:1.6940658945e-21
Apparent order of convergence at iteration 1:4.02658
Apparent order of convergence at iteration 2:4.21907
----------------------------------------
Known second derivative:
----------------------------------------
Iteration:0 value:1 error:1.0000000000e+00
Iteration:1 value:0.1086040047 error:1.0860400467e-01
Iteration:2 value:0.0001422922156 error:1.4229221563e-04
Iteration:3 value:-2.397593029e-13 error:2.3975930291e-13
Apparent order of convergence at iteration 1:3.98984
Apparent order of convergence at iteration 2:3.28069
```

Discussion: Convergence appears to be third order (or better) for both problems. In the third problem, we learned that the order of convergence of Newton's method depends on the second and third derivatives. Since we are dealing with a zero of multiplicity three ($f = 0,\ f' = 0,\ f'' = 0$), it seems justified to expect better than quadratic convergence.

Code:

```
function p6
  disp('----------------------------------------')
  disp('Known multiplicity:')
  disp('----------------------------------------')
  newton_times_m(1, @f, @fprime, 3, 0, 3);
  disp('----------------------------------------')
  disp('Known second derivative:')
  disp('----------------------------------------')
  newton_second_deriv(1, @f, @fprime, @f2prime, 3, 0);
end
```

```
function y = f(x)
  y = x*(1-cos(x));
end
function y = fprime(x)
  y = x*sin(x)-cos(x)+1;
end
function y = f2prime(x)
  y = 2*sin(x)+x*cos(x);
end

function newton_times_m(x, f, fprime, itcount, true, mult)
  disp(sprintf('Iteration:%d value:%.10g error:%.10e', 0, x, abs(x-true)))
  for i=1:itcount
    x = x - mult*f(x)/fprime(x);
    disp(sprintf('Iteration:%d value:%.10g error:%.10e', i, x, abs(x-true)))
    errors(i) = abs(x-true);
  end

  order_of_convergence(errors)
end

function newton_second_deriv(x, f, fprime, f2prime, itcount, true)
  disp(sprintf('Iteration:%d value:%.10g error:%.10e', 0, x, abs(x-true)))
  for i=1:itcount
    x = x - f(x)*fprime(x)/(fprime(x)^2-f(x)*f2prime(x));
    disp(sprintf('Iteration:%d value:%.10g error:%.10e', i, x, abs(x-true)))
    errors(i) = abs(x-true);
  end

  order_of_convergence(errors)
end

function order_of_convergence(seq)
  for at=1:length(seq)-1
    order = log(seq(at+1)) / log(seq(at+0));
    disp(sprintf('Apparent order of convergence at iteration %d:%g', at, order))
  end
end
```