Homework 6

Elizabeth Hunt

November 11, 2023

1 Question One

For $g(x) = x + f(x)$ then we know $g'(x) = 1 + 2x - 5$ and thus $|g'(x)|1$ is only true on the interval $(1.5, 2.5)$, and for $g(x) = x - f(x)$ then we know $g'(x) = 1 - (2x - 5)$ and thus $|g'(x)| < 1$ is only true on the interval $(2.5, 3.5)$.
Because we know the roots of f are $2, 3$ ($f(x) = (x - 2)(x - 3)$) then we can only be certain that $g(x) = x + f(x)$ will converge to the root 2 if we pick an initial guess between $(1.5, 2.5)$, and likewise for $g(x) = x - f(x)$, 3:

```
// tests/roots.t.c
UTEST(root, fixed_point_iteration_method) {
  // x^2 - 5x + 6 = (x - 3)(x - 2)
  double expect_x1 = 3.0;
  double expect_x2 = 2.0;

  double tolerance = 0.001;
  uint64_t max_iterations = 10;

  double x_0 = 1.55; // 1.5 < 1.55 < 2.5
  // g1(x) = x + f(x)
  double root1 =
      fixed_point_iteration_method(&f2, &g1, x_0, tolerance, max_iterations);
  EXPECT_NEAR(root1, expect_x2, tolerance);

  // g2(x) = x - f(x)
  x_0 = 3.4; // 2.5 < 3.4 < 3.5
  double root2 =
      fixed_point_iteration_method(&f2, &g2, x_0, tolerance, max_iterations);
  EXPECT_NEAR(root2, expect_x1, tolerance);
}
```

And by this method passing in `tests/roots.t.c` we know they converged within `tolerance` before 10 iterations.

2 Question Two

Yes, we showed that for $\epsilon = 1$ in Question One, we can converge upon a root in the range $(2.5, 3.5)$, and when $\epsilon = -1$ we can converge upon a root in the range $(1.5, 2.5)$.
See the above unit tests in Question One for each ϵ.

3 Question Three

See `test/roots.t.c -> UTEST(root, bisection_with_error_assumption)` and the software manual entry `bisect_find_root_with_error_assumption`.

4 Question Four

See `test/roots.t.c -> UTEST(root, fixed_point_newton_method)` and the software manual entry `fixed_point_newton_method`.

5 Question Five

See `test/roots.t.c -> UTEST(root, fixed_point_secant_method)` and the software manual entry `fixed_point_secant_method`.

6 Question Six

See `test/roots.t.c -> UTEST(root, fixed_point_bisection_secant_method)` and the software manual entry `fixed_point_bisection_secant_method`.

7 Question Seven

The existance of `test/roots.t.c`'s compilation into `dist/lizfcm.test` via `make` shows that the compiled `lizfcm.a` contains the root methods mentioned; a user could link the library and use them, as we do in Question Eight.

8 Question Eight

The given ODE $\frac{dP}{dt} = \alpha P - \beta P$ has a trivial solution by separation:

$$P(t) = Ce^{t(\alpha - \beta)}$$

And

$$P_0 = P(0) = Ce^0 = C$$

So $P(t) = P_0 e^{t(\alpha-\beta)}$.
We're trying to find t such that $P(t) = P_\infty$, thus we're finding roots of $P(t) - P_\infty$.
The following code (in `homeworks/hw_6_p_8.c`) produces this output:

```
$ gcc -I../inc/ -Wall hw_6_p_8.c ../lib/lizfcm.a -lm -o hw_6_p_8 && ./hw_6_p_8

a ~ 27.303411; P(27.303411) - P_infty = -0.000000
b ~ 40.957816; P(40.957816) - P_infty = -0.000000
c ~ 40.588827; P(40.588827) - P_infty = -0.000000
d ~ 483.611967; P(483.611967) - P_infty = -0.000000
e ~ 4.894274; P(4.894274) - P_infty = -0.000000
```

```
// compile & test w/
//  \--> gcc -I../inc/ -Wall hw_6_p_8.c ../lib/lizfcm.a -lm -o hw_6_p_8
//  \--> ./hw_6_p_8

#include "lizfcm.h"
#include <math.h>
#include <stdio.h>

double a(double t) {
  double alpha = 0.1;
  double beta = 0.001;
  double p_0 = 2;
  double p_infty = 29.85;

  return p_0 * exp(t * (alpha - beta)) - p_infty;
}

double b(double t) {
  double alpha = 0.1;
  double beta = 0.001;
  double p_0 = 2;
  double p_infty = 115.35;

  return p_0 * exp(t * (alpha - beta)) - p_infty;
}

double c(double t) {
  double alpha = 0.1;
  double beta = 0.0001;
  double p_0 = 2;
  double p_infty = 115.35;

  return p_0 * exp(t * (alpha - beta)) - p_infty;
}

double d(double t) {
  double alpha = 0.01;
  double beta = 0.001;
  double p_0 = 2;
  double p_infty = 155.346;

  return p_0 * exp(t * (alpha - beta)) - p_infty;
}

double e(double t) {
  double alpha = 0.1;
  double beta = 0.01;
  double p_0 = 100;
  double p_infty = 155.346;

  return p_0 * exp(t * (alpha - beta)) - p_infty;
```

```
}

int main() {
  uint64_t max_iterations = 1000;
  double tolerance = 0.0000001;

  Array_double *ivt_range = find_ivt_range(&a, -5.0, 3.0, 1000);
  double approx_a = fixed_point_secant_bisection_method(
      &a, ivt_range->data[0], ivt_range->data[1], tolerance, max_iterations);

  free_vector(ivt_range);
  ivt_range = find_ivt_range(&b, -5.0, 3.0, 1000);
  double approx_b = fixed_point_secant_bisection_method(
      &b, ivt_range->data[0], ivt_range->data[1], tolerance, max_iterations);

  free_vector(ivt_range);
  ivt_range = find_ivt_range(&c, -5.0, 3.0, 1000);
  double approx_c = fixed_point_secant_bisection_method(
      &c, ivt_range->data[0], ivt_range->data[1], tolerance, max_iterations);

  free_vector(ivt_range);
  ivt_range = find_ivt_range(&d, -5.0, 3.0, 1000);
  double approx_d = fixed_point_secant_bisection_method(
      &d, ivt_range->data[0], ivt_range->data[1], tolerance, max_iterations);

  free_vector(ivt_range);
  ivt_range = find_ivt_range(&e, -5.0, 3.0, 1000);
  double approx_e = fixed_point_secant_bisection_method(
      &e, ivt_range->data[0], ivt_range->data[1], tolerance, max_iterations);

  printf("a ~ %f; P(%f) = %f\n", approx_a, approx_a, a(approx_a));
  printf("b ~ %f; P(%f) = %f\n", approx_b, approx_b, b(approx_b));
  printf("c ~ %f; P(%f) = %f\n", approx_c, approx_c, c(approx_c));
  printf("d ~ %f; P(%f) = %f\n", approx_d, approx_d, d(approx_d));
  printf("e ~ %f; P(%f) = %f\n", approx_e, approx_e, e(approx_e));

  return 0;
}
```