Answer to Mathematical Experiment Exam 2017

Tony Xiang

May 30, 2019

1 Problem 1

```
x = [-2.0000 -1.6000 -1.2000 -0.8000 -0.4000 0 0.4000 0.8000
↪  1.2000 1.6000 2.0000];
y = [0.9820 0.9427 0.8320 0.5987 0.3100 0.1192 0.0392 0.0121
↪  0.0037 0.0011 0.0003];
f = @(c, x) c(1) ./ (1 + exp(c(2) + c(3) .* x));
options = optimset();
options.MaxFunEvals = 10000;
options.MaxIter = 10000;
options.TolX = 1e-10;
[c, resnorm, residual, exitflag, output] = lsqcurvefit(f, [1 1
↪  1], x, y, [], [], options);
c
```

The answers to problem 1 are the following:

$$\alpha = 1.0000$$
$$\beta = 2.0000$$
$$\gamma = 3.0000$$

2 Problem 2

```
func1 = @(x, y) exp(x.^2 + y.^2) + 2 * x.^2 .* y + sin(y.^4);
func2 = @(x) exp(x(1).^2 + x(2).^2) + 2 * x(1).^2 .* x(2) +
↪  sin(x(2).^4);

result1 = integral2(func1, -3 / 2, 3 / 2, @(x) (3 - sqrt(9 - 4 *
↪  x.^2)) / 2, @(x) (3 + sqrt(9 - 4 * x.^2)) / 2);
result2 = monte_carlo(func2, 2, 3000000, [-3 / 2, 0], [3 / 2,
↪  3], {@(x) x(1).^2 + x(2).^2 - 3 * x(2)});

function value = monte_carlo(funct, dimension, iteration, lb,
↪  ub, constraints)
```

```
    if nargin == 5
        constraints = {};
    end
    % monte-carlo main method
    points = zeros(dimension, iteration);
    total_value = 0;
    for i = 1: dimension
        points(i, :) = generate_random_points(iteration, lb(i),
         ↪  ub(i));
    end
    % tag for constraints
    for j = 1: iteration
        tag = true;
        for k = 1: size(constraints, 1) % check constraints
            if (constraints{k}(points(:, j)') > 0)
                tag = false;
                break;
            end
        end
        if (tag)
            total_value = total_value + funct(points(:, j)');
        end
    end
    value = total_value * prod(ub - lb) / iteration;
end

function point = generate_random_points(numbers, lb, ub)
    % random point generator by uniform distribution
    point = lb + (ub - lb) .* rand(1, numbers);
end
```

With an answer of $2.4838 \cdot 10^3$, compared to the MatLab built-in integral function of $2.4831 \cdot 10^3$.

Please note that a **coordinate transformation** should be performed before you integrate this problem, or a proper area of integration is needed.

3 Problem 3

```
% Runge-Kutta solution
options = odeset('RelTol',1e-12, 'AbsTol', [1e-10 1e-10 1e-10]);
[t, coord] = ode45(@(t, c)rigid(t, c, 15), [0 2.3732], [0 0 0],
↪  options);

plot3(coord(:, 1), coord(:, 2), coord(:, 3), 'linewidth', 1);
xlabel('x-axis');
```

```
ylabel('y-axis');
zlabel('z-axis');
set(gca, 'fontsize', 16);

ppx = spline(t, coord(:, 1));
ppy = spline(t, coord(:, 2));
ppz = spline(t, coord(:, 3));

distance = @(t) sqrt((10 + 2 * cos(t) - fnval(ppx, t)).^2 + ...
                     (20 + 4 * sin(t) - fnval(ppy, t)).^2 + ...
                     (30 + 5 * sin(2 * t) - fnval(ppz, t)).^2);

solution = fsolve(distance, 10);
solution

function dc = rigid(t, c, w)
    dc = zeros(3, 1);
    dc(1) = w ./ sqrt((10 + 2 * cos(t) - c(1)).^2 + ...
                      (20 + 4 * sin(t) - c(2)).^2 + ...
                      (30 + 5 * sin(2 * t) - c(3)).^2) .* (10 +
                      ↪  2 * cos(t) - c(1));
    dc(2) = w ./ sqrt((10 + 2 * cos(t) - c(1)).^2 + ...
                      (20 + 4 * sin(t) - c(2)).^2 + ...
                      (30 + 5 * sin(2 * t) - c(3)).^2) .* (20 +
                      ↪  4 * sin(t) - c(2));
    dc(3) = w ./ sqrt((10 + 2 * cos(t) - c(1)).^2 + ...
                      (20 + 4 * sin(t) - c(2)).^2 + ...
                      (30 + 5 * sin(2 * t) - c(3)).^2) .* (30 +
                      ↪  5 * sin(2 * t) - c(3));
end
```

The chasing plot is:

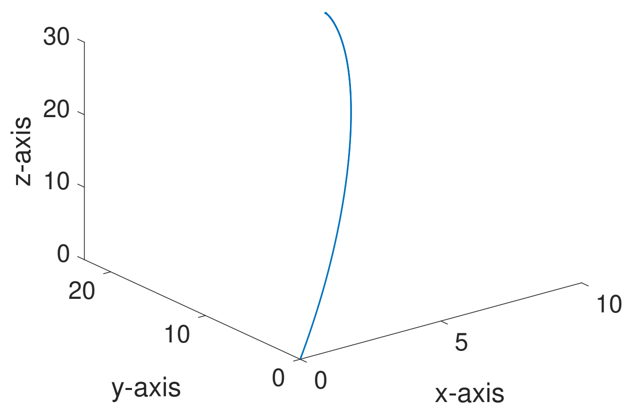

Figure 1: Chasing Plot

The answer is approximately 2.3732 seconds.

Please note that using normal ode45 function will only get a approximation of that solution. Other methods like spline interpolation should be performed after solving the ODE to enhance the precision.

4 Problem 4

```
phi = @(x) ((-1 / 2 <= x) & (x < 1 / 2));
for i = 1: 10
    phi = @(x) (-(1 + sqrt(3)) * phi(2 * x) + (3 + sqrt(3)) *
    ↪  phi(2 * x - 1) + ...
          (3 - sqrt(3)) * phi(2 * x - 2) + (1 - sqrt(3)) * phi(2
          ↪  * x - 3)) / 4;
end

fplot(phi, [0, 3], 'linewidth', 1);
```

The plot of that function is:

Please note that this problem is REALLY time-consuming. You can refer to wavelet when finishing this problem.

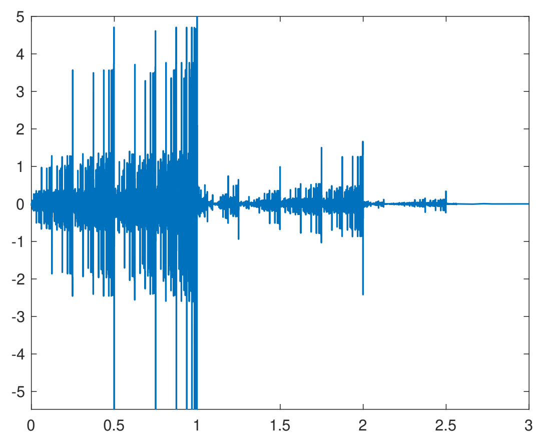

Figure 2: Plot of the function