PHY5340 Laboratory Report 2

Table of Contents

Jeremiah O'Neil, SN6498391

Question 1: Root Finding

Rootfinding methods:

```
type('bisect.m')
type('NR.m')
type('secant_NR.m')
```

```
function [root, est_err, numiter] = bisect(maxiter, tol, func, a, b)
% Based on C1_1.m
assert(func(a)*func(b) < 0);
est_err = 1; numiter = 0;
while est_err > tol
    numiter = numiter + 1;
    mid = (a + b) / 2;
    if func(mid)*func(a) < 0
        b = mid;
    elseif func(mid)*func(b) < 0
        a = mid;
    end
    est_err = abs(a - b) / mid;
    if numiter > maxiter, break, end
end
root = mid;
end

function [root, est_err, numiter] = NR(maxiter, tol, func, dfunc, a)
est_err = 1; numiter = 0;
x = a;
while est_err > tol
    numiter = numiter + 1;
    xprev = x;
    x = xprev - func(xprev) / dfunc(xprev);
    est_err = abs((x - xprev) / x);
    if numiter > maxiter, break, end
end
root = x;
end

function [root, est_err, numiter] = secant_NR(maxiter, tol, func, a1,
 a0)
```

```
est_err = 1; numiter = 0;
xprev = a0; x = a1;
while est_err > tol
    numiter = numiter + 1;
    xnext = x - func(x) * (x - xprev) / (func(x) - func(xprev));
    xprev = x; x = xnext;
    est_err = abs((x - xprev) / x);
    if numiter > maxiter, break, end
end
root = x;
end
```

Numerical extremum classifier:

```
type('check_extremum.m')
```

```
function [open_dir] = check_extremum(func, r, delta)
sgn_diff_l = sign(func(r) - func(r - delta));
sgn_diff_r = sign(func(r) - func(r + delta));
if (sgn_diff_l * sgn_diff_r) ~= 1, open_dir = 0; return, end
open_dir = sgn_diff_l;
end
```

Potential function and its derivatives:

```
type('NaCl_pot.m')
type('NaCl_force.m')
type('NaCl_stiffness.m')
```

```
function [pot] = NaCl_pot(r)
% NaCl interatomic potential (in eV) at separation r (in angstrom)
pot = 1090 * exp(-r ./ 0.33) - 14.4 ./ r;
end

function [force] = NaCl_force(r)
% NaCl interatomic force (in eV/angstrom) at separation r (in
 angstrom)
force = 14.4 ./ r.^2 - 3303 * exp(-r ./ 0.33);
end

function [stiffness] = NaCl_stiffness(r)
% NaCl interatomic stiffness (eV/angstrom^2) at separation r
 (angstrom)
stiffness = 10009 * exp(-r ./ 0.33) - 28.8 ./ r.^3;
end
```

Execute!

```
type('Lab2Q1.m')
Lab2Q1
```

```
tol = 1E-6;
%% Bisection
```

```
disp(' ')
disp('Solving by bisection...')
[eq_pos, est_err, numiter] = bisect(50, tol, @(r)NaCl_force(r), 0.1,
 15)
if check_extremum(@(r)NaCl_pot(r), eq_pos, 10*tol) == -1
    disp('This is a point of minimum potential.')
end
%% Newton-Raphson
disp(' ')
disp('Solving by the Newton-Raphson method...')
[eq_pos, est_err, numiter] = NR(50, tol, @(r)NaCl_force(r),...
                                @(r)NaCl_stiffness(r), 1)
if check_extremum(@(r)NaCl_pot(r), eq_pos, 10*tol) == -1
    disp('This is a point of minimum potential.')
end
%% Secant
disp(' ')
disp('Solving by the secant method...')
[eq_pos, est_err, numiter] = secant_NR(50, tol, @(r)NaCl_force(r), 1,
 0.5)
if check_extremum(@(r)NaCl_pot(r), eq_pos, 10*tol) == -1
    disp('This is a point of minimum potential.')
end
```

```
Solving by bisection...

eq_pos =

    2.3605


est_err =

   7.5246e-07


numiter =

    23

This is a point of minimum potential.

Solving by the Newton-Raphson method...

eq_pos =

    2.3605


est_err =

   7.9155e-09
```

```
numiter =

     9

This is a point of minimum potential.

Solving by the secant method...

eq_pos =

    2.3605


est_err =

   1.4932e-09


numiter =

    13

This is a point of minimum potential.
```

The choice of initial conditions in the use of these methods has a significant impact on convergence. Bisection will fail outright unless there is an odd number of zero-crossings between the limits specified. Since each iteration of the bisection algorithm halves the search domain the number of requisite iterations reliably increases with the binary logarithm of (b - a). The NR method is guided by the derivative of the function, and may fail to converge for functions which are badly behaved (noisy in a spectral sense, I suppose) or whose gradient in the area around the initial position points away from the root(s); for the NaCl interatomic force function NR converges very quickly with a good start, but it fails to converge (runs away) for a bad one (ie., one out on the gaussian tail). The secant method is sensitive to the initial postion in the same way as the NR method, but is additionally sensitive to the chosen initial "previous" point: a large difference between these initial conditions can slingshot the search away from the start in a way that has little to do with the function, and too small a difference can lead to a slow start. In my tests the secant method was never better than, but for certain ICs could be as good as, the NR method. All methods could converge to the wrong root if given poor ICs. In general, for ICs which avoided the root near the origin, bisection converged reliably and in a reliable number of iterations while the NR and secant methods converged more quickly than bisection when they succeeded but required ICs close to the root to succeed at all.

Question 2: Solving Linear Systems

LU factorization method:

```
type('LU_naive.m')
```

```
function [L, U] = LU_naive(A)
assert(size(A, 1) == size(A, 2))
L = eye(size(A));
U = zeros(size(A)); U(1, :) = A(1, :);
for n = 1:(size(A, 1) - 1) % nth diag elem, starting target U col and
 L row
```

```
    % Note, 1:0 is 'empty': it indexes to 'empty' and operates as
 zero.
    L(n:end, n) = (A(n:end, n) - L(n:end, 1:n-1) * U(1:n-1, n)) / U(n,
 n);
    U(n+1, n+1:end) = A(n+1, n+1:end) - L(n+1, 1:n) * U(1:n, n+1:end);
end
end
```

LU decomposition linear system solver:

```
type('solve_linsys_LU.m')
```

```
function [x, y, L, U] = solve_linsys_LU(A, b)
assert(size(A, 1) == size(A, 2) && size(A, 1) == size(b, 1))
N = size(b, 1);
y = zeros(N, 1); x = zeros(N, 1);
[L, U] = LU_naive(A);
for i = 1:N
    y(i) = b(i) - L(i, 1:i-1) * y(1:i-1);
end
for i = N - (0:N-1)
    x(i) = 1/U(i, i) * (y(i) - U(i, i+1:N) * x(i+1:N));
end
end
```

Input generator:

```
type('problem_of_size.m')
```

```
function [b, A, H, a] = problem_of_size(N)
a = linspace(1, 3, N);
b = [1; zeros(N-1, 1)];
H = toeplitz(exp(-a));
A = expm(-1i*H);
end
```

Execute!

```
type('Lab2Q2.m')
Lab2Q2
```

```
%% Solution for N = 10
[b, A] = problem_of_size(10);
x = solve_linsys_LU(A, b)
%% Benchmark for N = 500
[b, A] = problem_of_size(500);
disp('Solving with my implementation...')
tic; x = solve_linsys_LU(A, b); toc
disp('Solving with the backslash operator...')
tic; x = A\b; toc

x =
```

```
   0.8541 + 0.2853i
  -0.1555 + 0.2003i
  -0.1526 + 0.1347i
  -0.1424 + 0.0853i
  -0.1282 + 0.0490i
  -0.1124 + 0.0234i
  -0.0965 + 0.0062i
  -0.0814 - 0.0043i
  -0.0676 - 0.0095i
  -0.0553 - 0.0107i

Solving with my implementation...
Elapsed time is 0.196605 seconds.
Solving with the backslash operator...
Elapsed time is 0.010487 seconds.
```

Published with MATLAB® R2016a