Math 164 Homework 6

Nick Monozon

Problem 7.5

Proof. Let $P(k)$ be the statement "$F_{k-2}F_{k+1} - F_{k-1}F_k = (-1)^k$." We will first show that the base case is satisfied, that is, $P(2)$ holds. We have that

$$\begin{aligned} F_0F_3 - F_1F_2 &= (1)(3) - (1)(2) \\ &= 3 - 2 \\ &= (-1)^2. \end{aligned}$$

The base case therefore holds. We will now proceed to the inductive step. Assume that $P(k)$ holds. We want to show that $P(k+1)$ also holds where $k \geq 2$. Thus, using the recursive definition of the Fibonacci sequence, we have that

$$\begin{aligned} F_{k-1}F_{k+2} - F_kF_{k+1} &= F_{k-1}(F_k + F_{k+1}) - (F_{k-2} + F_{k-1})F_{k+1} \\ &= F_{k-1}F_k + F_{k-1}F_{k+1} - F_{k-2}F_{k+1} - F_{k-1}F_{k+1} \\ &= F_{k-1}F_k - F_{k-2}F_{k+1} \\ &= -1 \times (-1)^k \\ &= (-1)^{k+1}. \end{aligned}$$

We have thus shown that $P(k+1)$ is true. By the Principle of Mathematical Induction, $P(k)$ is true for $k \geq 2$. □

Problem 7.6

Let $a_k = F_k$ and $b_k = F_{k-1}$. We have that

$$\begin{bmatrix} a_{k+1} \\ b_{k+1} \end{bmatrix} = \begin{bmatrix} 1 & 1 \\ 1 & 0 \end{bmatrix} \begin{bmatrix} a_k \\ b_k \end{bmatrix}.$$

subject to the initial condition

$$\begin{bmatrix} a_k \\ b_k \end{bmatrix} = \begin{bmatrix} 1 \\ 0 \end{bmatrix}.$$

Let

$$\boldsymbol{M} = \begin{bmatrix} 1 & 1 \\ 1 & 0 \end{bmatrix}.$$

We see that

$$F_n = a_n = [1, 0] \begin{bmatrix} a_n \\ b_n \end{bmatrix} = [1, 0]\boldsymbol{M}^n \begin{bmatrix} 1 \\ 0 \end{bmatrix}.$$

Since $\boldsymbol{M}$ is a symmetric matrix and therefore diagonalizable, we can express it as

$$\boldsymbol{M} = \begin{bmatrix} \boldsymbol{v}^\top \\ \boldsymbol{w}^\top \end{bmatrix} \begin{bmatrix} \lambda_1 & 0 \\ 0 & \lambda_2 \end{bmatrix} \begin{bmatrix} \boldsymbol{v} & \boldsymbol{w} \end{bmatrix}$$

where $\lambda_1 = (1 + \sqrt{5})/2$ and $\lambda_2 = (1 - \sqrt{5})/2$ are the eigenvalues of $\boldsymbol{M}$. We correspondingly find that

$$\boldsymbol{v} = -\frac{1}{5^{1/4}} \begin{bmatrix} \sqrt{2/(\sqrt{5} - 1)} \\ \sqrt{(\sqrt{5} - 1)/2} \end{bmatrix}$$

and

$$w = -\frac{1}{5^{1/4}} \begin{bmatrix} \sqrt{(\sqrt{5}-1)/2} \\ -\sqrt{2/(\sqrt{5}-1).} \end{bmatrix}$$

It therefore follows that

$$\begin{aligned} F_n &= v^\top \begin{bmatrix} \lambda_1 & 0 \\ 0 & \lambda_2 \end{bmatrix}^n v \\ &= \lambda_1^n v_1^2 + \lambda_2^n v_2^2 \\ &= \frac{1}{\sqrt{5}} \left(\left(\frac{1+\sqrt{5}}{2} \right)^{n+1} - \left(\frac{1-\sqrt{5}}{2} \right)^{n+1} \right). \end{aligned}$$

Problem 7.7

The number $\ln 2$ is the root of the equation $f(x) = e^x - 2$. As such, applying Newton's Method to this root-finding problem, we have

$$x^{(k+1)} = x^{(k)} - \frac{e^{x^{(k)}} - 2}{e^{x^{(k)}}}$$

for $k \geq 0$. Performing 2 iterations of Newton's Method for an initial guess of $x^{(0)} = 1$ with the attached MATLAB script (see Appendix A), we obtain approximations $x^{(1)} = 0.7357588823$ and $x^{(2)} = 0.6940422999$.

Problem 7.8

(a) We first compute $g'(x) = 2e^x/(e^x+1)^2$. Hence, the algorithm for Newton's Method is

$$\begin{aligned} x^{(k+1)} &= x^{(k)} - \frac{(e^{x^{(k)}} - 1)/(e^{x^{(k)}} + 1)}{2e^{x^{(k)}}/(e^{x^{(k)}} + 1)^2} \\ &= x^{(k)} - \frac{e^{2x^{(k)}} - 1}{2e^{x^{(k)}}} \\ &= x^{(k)} - \sinh x^{(k)} \end{aligned}$$

for $k \geq 0$.

(b) For the algorithm to cycle, we require that $x^{(1)} = -x^{(0)}$, which is clear by symmetry. This means that $x^{(0)}$ must satisfy

$$-x^{(0)} = x^{(0)} - \sinh x^{(0)}.$$

By rearranging this expression, we find that the algorithm cycles for $x^{(0)} = k$, where k is the solution to $2k = \sinh k$.

(c) Using the result from part (b), we have that the algorithm cycles for $x^{(0)} \in (-k, k)$, where k is again the solution to $2k = \sinh k$. We motivate this argument by symmetry as before.

Problem 7.9

The quadratic function that agrees with the given points $\left(x^{(k-i)}, f(x^{(k-i)})\right)$ for each $i = 0, 1, 2$ satisfy the following 3 quadratic formulas:

$$\begin{aligned} a(x^{(k)})^2 + bx^{(k)} + c &= f(x^{(k)}), \\ a(x^{(k-1)})^2 + bx^{(k-1)} + c &= f(x^{(k-1)}), \\ a(x^{(k-2)})^2 + bx^{(k-2)} + c &= f(x^{(k-2)}), \end{aligned}$$

where we solve for a, b, c using this system of equations. Since these are quadratic formulas, we have by the axis of symmetry that $x^{(k+1)} = -b/2a$. From this, we can find a and b to get that

$$x^{(k+1)} = \frac{\sigma_{12} f(x^{(k)}) + \sigma_{20} f(x^{(k-1)}) + \sigma_{01} f(x^{(k-2)})}{2(\delta_{12} f(x^{(k)}) + \delta_{20} f(x^{(k-1)}) + \delta_{01} f(x^{(k-2)}))}$$

where we let $\sigma_{ij} = (x^{(k-i)})^2 - (x^{(k-j)})^2$ and $\delta_{ij} = x^{(k-i)} - x^{(k-j)}$.

Problem 7.10

(a) The code below provides a MATLAB implementation of the Secant Method (`secant_method.m`). The corresponding `.m` file can be be found on my GitHub, @nickmonozon.

```
%% Secant Method

% This code approximates a solution to f(x) = 0 given initial approximations p0 and p1.

% INPUTS:
%   pn1        =   initial approximation
%   p1         =   another initial approximation
%   epsilon    =   tolerance

% OUTPUTS:
%   p          =  approximate solution

%% Information and set up

f = @(x) (2*x-1)^2 + 4*(4-1024*x)^4;        % function whose root we want to approximate
pn1 = 0;                                    % first initial approximation
p0 = 1;                                     % second initial approximation
epsilon = 1e-5;                             % tolerance, e.g. 1e-4 = 10^{-4}


%% Secant Method

qn1 = f(pn1);
q0 = f(p0);

while true

    % get p_i
    p = p0 - ((p0 - pn1) * q0)/(q0 - qn1);

    % check stopping condition
    if(abs(p - p0) < abs(p0)*epsilon)
        break;
    end


    % prepare for next iteration
    pn1 = p0;
    qn1 = q0;
    p0 = p;
    q0 = f(p);

end

%% Display Information

fprintf('\nSecant Method approximated the solution p = %.10f.\n\n',p);
```

(b) We want to find the "root" of the equation $g(x) = (2x-1)^2 = 4(4-1024x)^4$ with initial guesses $x^{(-1)} = 0$ and $x^{(0)} = 1$ with tolerance $\epsilon = 10^{-5}$. Using `secant_method.m`, we obtain an approximation of $p = 0.0038664095$, where $g(p) = 0.9846$. Clearly, this is not a root, but it may be a minimizer. In fact, the function $g(x)$ actually has no root. To formalize this, we have that $(2x-1)^2 \geq 0$ for all $x \in \mathbb{R}$, but $(2x-1)^2 = 0$ only for $x = 1/2$. Additionally, $(4-1024x)^4 \geq 0$ for all $x \in \mathbb{R}$, but

$(4-1024x)^4 = 0$ only when $x = 4/1024 = 1/256$. Since both values of x cannot be obtained simultaneously, we conclude that $g(x)$ has no roots.

Problem 7.11

The following code provides a MATLAB implementation of the line search algorithm using the Secant Method. Note that we also need a file `grad.m` with contains the gradient. This code can also be found on my GitHub, @nickmonozon.

```
%% Line search using the Secant Method

% Information and set up

% INPUTS:
%   'grad'      =   .m file with gradient
%   x           =   starting line search point
%   d           =   search direction
%   epsilon     =   tolerance, e.g. 1e-4 = 10^{-4}
%   max         =   maximum number of iterations

% OUTPUTS:
%   alpha       =   value returned by function

alpha_curr = 0;
alpha = 0.001;
dphi_zero = feval(grad,x)'*d;
dphi_curr=dphi_zero;
epsilon = 1e-4;
max = 100;

i = 0;

while abs(dphi_curr) > epsilon*abs(dphi_zero),
    alpha_o = alpha_curr;
    alpha_curr = alpha;
    dphi_o = dphi_curr;
    dphi_curr = feval(grad, x+alpha_curr*d)'*d;
    alpha = (dphi_curr*alpha_o-dphi_o*alpha_curr)/(dphi_curr-dphi_o);
    i = i + 1;
    if (i >= max) & (abs(dphi_curr)>epsilon*abs(dphi_zero)),
        fprintf('\nLine search algorithm converges after %d iterations.\n\n',i);
        break;
    end
end
```

Appendix A

The code below provides a MATLAB implementation of Newton's Method (`newtons_method.m`). The `.m` file can be found on my GitHub, @nickmonozon.

```
%% Newton's Method

% This code approximates a solution to f(x) = 0 given an initial
% approximation p0.

% INPUTS:
%   p0          = initial approximation
%   epsilon     = tolerance
```

```
%   max_iter    = maximum number of iterations

% OUTPUTS:
%   p           =  approximate solution
%   or error message

%% Information and set up

f = @(x) exp(x) - 2;        % function whose root we want to approximate
fprime = @(x) exp(x);       % derivative of function whose root we want to approximate

p0 = 1;

tol = 1e-4;                 % tolerance, 1e-4 = 10^{-4}

max_iter = 30;              % max number of iterations

%% Newton's Method

i = 1;                                          % iteration count

fprintf('i\tp_i\t\tf(p_i)\n');                  % for display
fprintf('%d\t%.10f\t%.10f\n',0,p0,f(p0));       % displays iteration i, p_i, f(p_i)

while( i <= max_iter)

    % get p_i
    p = p0 - f(p0)/fprime(p0);    % p_i

    % display information
    fprintf('%d\t%.10f\t%.10f\n',i,p,f(p));   % displays iteration i, p_i, f(p_i)

    % check stopping condition
    if(abs(p - p0) < epsilon)
        break;
    end

    % increase iteration count
    i = i + 1;

    % prepare for next iteration
    p0 = p;

end

%% Display Information

if( i <= max_iter )
    fprintf('\nNewton''s Method approximated the solution p = %.10f after %d iterations.\n\n',p,i);
else
    fprintf('\nNewton''s Method did not converge within the tolerance in %d iterations.\n\n',max_iter)
end
```