CONTENTS

I Problem Outline 2

II A General Solution 2

II-A The initial subroutine 2

II-B The update subroutine 2

II-C The updateTerms subroutine 4

II-D The updateAnswer subroutine 4

III Particular Solutions 5

III-A A particularly general mainline program 5

List of Flowcharts

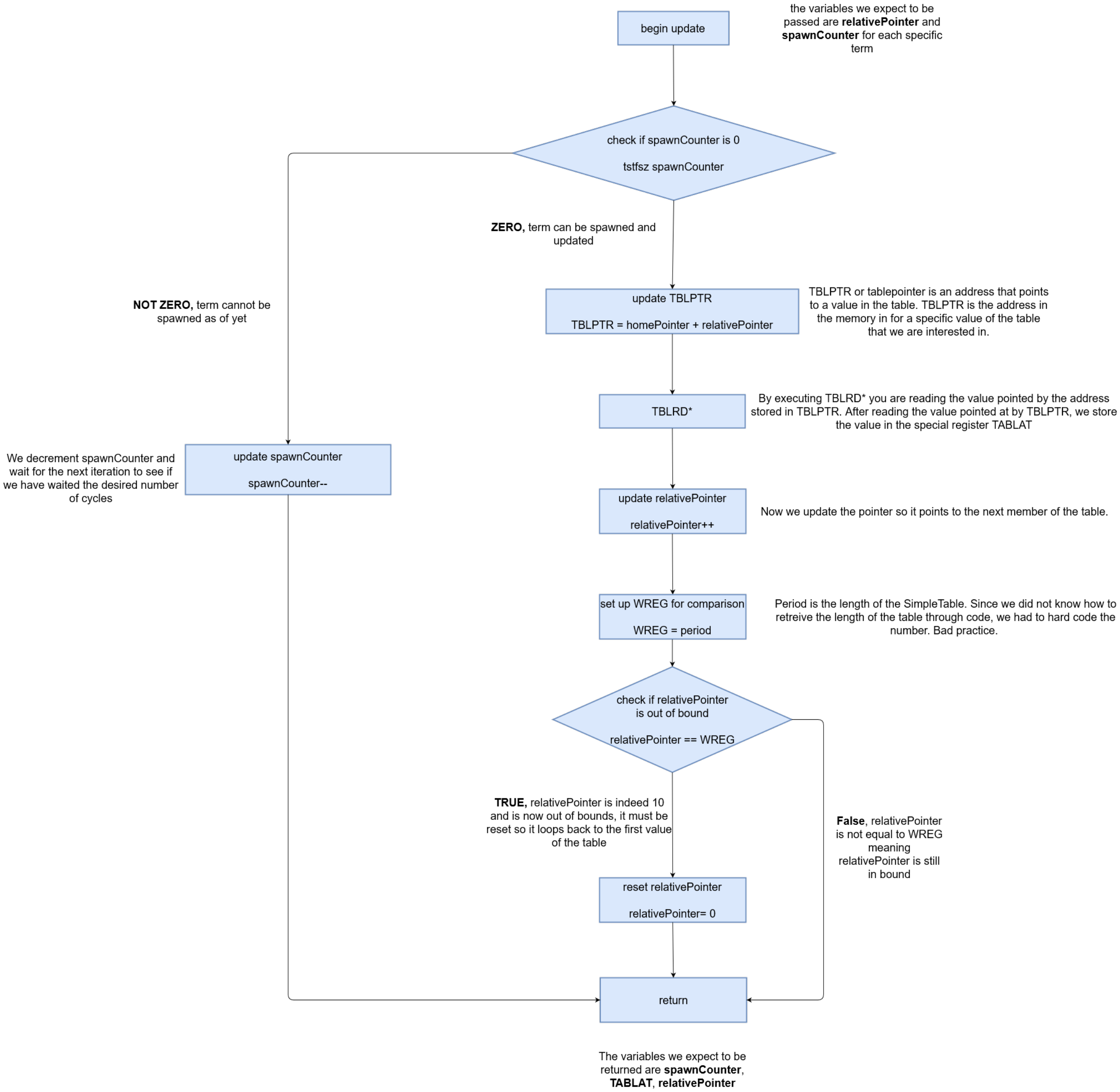

Fig. 1. Update subroutine flowchart

Exp 6 - Discrete-Time Series Averaging Filter Part II

Anas Ashraf

I. Problem Outline

This is an extension of Exp 5 - Discrete-Time Series Averaging Filters and thus, using the same notation and conditions that we did in Exp 5 we wish to solve the general problem:

$$y[n] = \frac{x[n] + x[n-k_1] + x[n-k_2] + x[n-k_3]}{4} \quad \forall n : n \in \mathbb{N} \tag{1}$$

Such that $k_1, k_2, k_3 \in \mathbb{N}$ and just like in Exp 5, the size of array y is infinite and array x is finite. If array x only has N entries and $\gamma > N$, then $x[\gamma]$ is understood to be by $x[\gamma \ mod \ N]$ when appropriate. The only meaningful way in which this is an extension of Exp 5 is that now we consider four terms of $x[n]$ when computing $y[n]$ instead of two terms like previously.

II. A General Solution

A. The initial subroutine

The only important thing we do in the initial subroutine is initialize the values of the table into the memory as shown in figure 2. We define the variable homePointer to be the memory address which points to the first value in the array just like we did for experiment 5.

```
;Think of this as initializing the table values and loading them into memory
MOVLF upper SimpleTable,TBLPTRU
MOVLF high  SimpleTable,TBLPTRH
MOVLF low   SimpleTable,TBLPTRL

;This declares constant homePointer which is the address of the first
;entry in the SimpleTabl, in this specific example we have
;homePointer = 190 but this solution is more robust by not defining it
;explicitly

copyRegister homePointer, TBLPTR    ;;homePointer = TBLPTR
```

Fig. 2. Code for the Initial subroutine

B. The update subroutine

Since this experiment is a trivial extension of experiment 5 we see to it that the solution must also be a trivial extension. Let us call each term in equation 1 that is in the numerator as simply terms. Thus, each term can be read from an arbitrary table for any value of k (which we will also call spawnCounter) as shown in listing 1. For an arbitrary array we pass the relativePointer and spawnCounter which is unique for each terms since each term spawns at different times and points to a different value in the array. If the spawnCounter or k value is not zero, which is for any term with a k in it, then we pass into the first if statement and decrement spawnCounter (which recall is unique for each term) and return the values spawnCounter, TABLAT (which in this instance would be 0), and relativePointer.

If however, the spawnCounter is zero (like the first term) we skip the if statement and update the tablePointer to be the sum of homePointer (which we could define in an intitial subroutine to be the memory address of the first entry of the array) and the relativePointer(which is unique to each term). We then define TABLAT to be the value stored in the memory address of tablePointer and increment relativePointer. Then we check to see if the relativePointer is still in bounds by checking to see if the relativePointer is equal to the period. The period is the length of the array which we could define in the iniial subroutine. If the relativePointer is indeed equivalent to the period then we reset the relativePointer. Finally, we return the values spawnCounter, TABLAT (now updated), and relativePointer (unique for each term).

Now we translate the pseudocode shown in listing 1 to the flowchart shown in figure 1. Before we begin the flowchart, we assume like we did for the code listing that we have passed two inputs, being relativePointer and spawnCounter. We first

```
# This pseudo-code is for reading an array and storing the values in some arbitrary variable

array = [180,240]   //arbitrary array

def update(relativePointer, spawnCounter):

    if(spawnCounter !=0):
        spawnCounter = spawnCounter -1
        return spawnCounter, TABLAT, relativePointer

    tablePointer = homePointer + relativePointer    #homePointer defined in initialize subroutine
    TABLAT = array[tablePointer]
    relativePointer++

    if(relativePointer == period):  #period defined in mainline
        relativePointer = 0
    return spawnCounter, TABLAT, relativePointer
```

Listing 1: Pseudocode for updating arbitrary term

```
;;;;;;;;;update subroutine;;;;;;;;;;;;;;;;;;;;;;;

update      ;the arguments passed are relativePointer and spawnCounter
    tstfsz spawnCounter         ;checks if time to spawn and update the register has arrived
    bra updateSpawnCount            ;not zero, not yet time to spawn or update register

    updateNoCounter
    ;Here tortoiseSpawnCounter is indeed zero, time to spawn tortoise and update value
    addTwoRegisters TBLPTR, homePointer, relativePointer ;TBLPTR = homePointer + relativePointer
    TBLRD*
    incf relativePointer, F
;   copyRegister tortoiseValue, TABLAT      moving this to mainline

    copyRegister WREG, period       ;The period is the length of SimpleTable
    cpfseq relativePointer          ;checks if relativePointer == period
    return                  ;relativePointer != period

    ;relativePointer = period
    MOVLF 0, relativePointer
    return

        ;updateSpawnCounter catch
        updateSpawnCount
        decf spawnCounter, F
return      ;return spawnCounter, relativePointer, TABLAT
```

Fig. 3. Code for the update subroutine in implementing a general solution

check to see if the spawnCounter is zero and if that is not true we decrement the spawnCounter and return the variables spawnCounter, relativePointer, and TABLAT.

If spawnCounter is indeed zero we update TBLPTR by assigning it to be the result of the homePointer (defined in the initial subroutine to be the memory which points to the first value of the array) and the relativePointer (unique for each term). We then read the table using TBLPTR and store that value in TABLAT. We then update relativePointer and set the WREG to be the period to prepare for an out of boudns check. The period is defined to be the length of the array we are dealing with. Now we check to see if relativePointer is indeed equal to the period and if it is not we simply return the three variables TABLAT, relativePointer, and spawnCounter. If relativePointer is indeed equal to the period we simply just reset the relativePointer to be zero. The code representation for this can be seen in figure 3. Notice the code is simply an extension of the code presented in experiment 5, specifically of the updateTortoise subroutine. It is a trivial extension.

C. The updateTerms subroutine

This subroutine shown in figure 4 is very simple. It passes the variables of spawnCounter and relativePointer (that are unique to each term as a global variable) into the update subroutine. Then it copies the outputs which are relativePointer, TABLAT, and spawnCounter(if relevant). We also sanitize the TABLAT so it does not spill over to other terms. Notice the essential logic is the same for each code block, it is simply copied and pasted with slight alterations so as to be unique for each term. The first term does not have a spawnCounter so we skip the check to see if spawnCounter is zero yet, skipping to updateNoCounter subroutine which is simply the update subroutine but without the spawnCounter check shown in figure 3. This is very quick and dirty code.

```
;;;;;;;;;;;;;;;;updateTerms subroutine;;;;;;;;;;;;;;;;;;;;;;;;;
updateTerms

    copyRegister relativePointer, firstPointer  ;relativePointer is local variable
    rcall updateNoCounter               ;update subroutine updates spawnCounter, TABLAT, relativePointer
    copyRegister firstPointer, relativePointer  ;firstPointer is global variable
    copyRegister firstTerm, TABLAT          ;updates first term with TABLAT that was received
    MOVLF 0, TABLAT                         ;sanitize TABLAT

    copyRegister spawnCounter, offsetOne        ;spawnCounter is local variable
    copyRegister relativePointer, secondPointer     ;relativePointer is local variable
    rcall update                    ;update subroutine updates spawnCounter, TABLAT, relativePointer
    copyRegister offsetOne, spawnCounter        ;offsetOne is global variable
    copyRegister secondPointer, relativePointer ;firstPointer is global variable
    copyRegister secondTerm, TABLAT         ;updates first term with TABLAT that was received
    MOVLF 0, TABLAT                         ;sanitize TABLAT

    copyRegister spawnCounter, offsetTwo        ;spawnCounter is local variable
    copyRegister relativePointer, thirdPointer  ;relativePointer is local variable
    rcall update                    ;update subroutine updates spawnCounter, TABLAT, relativePointer
    copyRegister offsetTwo, spawnCounter        ;offsetOne is global variable
    copyRegister thirdPointer, relativePointer  ;firstPointer is global variable
    copyRegister thirdTerm, TABLAT          ;updates first term with TABLAT that was received
    MOVLF 0, TABLAT                         ;sanitize TABLAT

    copyRegister spawnCounter, offsetThree      ;spawnCounter is local variable
    copyRegister relativePointer, fourthPointer ;relativePointer is local variable
    rcall update                    ;update subroutine updates spawnCounter, TABLAT, relativePointer
    copyRegister offsetThree, spawnCounter      ;offsetOne is global variable
    copyRegister fourthPointer, relativePointer ;firstPointer is global variable
    copyRegister fourthTerm, TABLAT         ;updates first term with TABLAT that was received
    MOVLF 0, TABLAT                         ;sanitize TABLAT


return
```

Fig. 4. Code for the udpateTerms subroutine

D. The updateAnswer subroutine

The updateAnswer subroutine shown in figure 5 can be better understood by breaking down equation 1 as the being rewritten as as equation 2.

$$\frac{1}{2}\left(\frac{x[n]+x[n-k_1]}{2}+\frac{x[n-k_2]+x[n-k_3]}{2}\right) \tag{2}$$

In figure 5, we begin by first computing the sum of $x[n]$ and $x[n-k_1]$ or the sum of the first and the second term and store it in answerOne. We then update answerOne by dividing it by two through a right rotation. We then continue by computing the sum of $x[n-k_2]$ and $x[n-k_3]$ and storing it in answerTwo. We then update answerTwo by dividing it by two. We then add answerOne and answerTwo storing it in answer. We then update answer by dividng it by two. This yields us the entry in $y[n]$ for some n. With this our general solution is complete and we can begin implenting a particular set of solutions.

```
;;;;;;;;;;;;update Answer;;;;;;;;;;;;;;;;;;;;;;;;

updateAnswer
    addTwoRegisters answerOne, firstTerm, secondTerm    ;answerOne = firstTerm + secondTerm
    rrcf answerOne,F                    ;answerOne = answerOne /2

    addTwoRegisters answerTwo, thirdTerm, fourthTerm    ;answerTwo = thirdTerm + fourthTerm
    rrcf answerTwo, F                   ;answerTwo = answerTwo /2

    addTwoRegisters answer, answerOne, answerTwo        ;answerOne = answerOne + answerTwo
    rrcf answer,F                       ;answerOne = answerOne /2
    return
```

Fig. 5. Code for the updateAnswer subroutine

III. Particular Solutions

A. A particularly general mainline program

The mainline program shown in figure 6 can be easily changed for any desired particular solution. The values of $k_1, k_2, and k_3$ can be set by changing offsetOne, offsetTwo, and offsetThree respectively. Finally, we must hardcode the value of the period depending on what we choose the SimpleTable to be. Based on our selection of the simpleTable by either commenting/uncommenting certain sections, we will have different periods of the table. In the mainloop we simply run through the updateTerms and updateAnswer subroutine over and over. Thus we can change the values of k_1, k_2, k_3 to be either $\{1, 2, 3\}$, $\{2, 4, 6\}$, $\{3, 6, 9\}$, or any three integers we choose (we chose the arbitrary indices to be $\{1, 3, 5\}$). The project project outlines states that the three integers must be less than 15 but our implementation has no such restriction. An exhaustive set of values for all $y[n]$ for any the given $x[n]$ and the given k_1, k_2, k_3 is shown in tables I-IV.

```
;;;;;;; Mainline program ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

Mainline
        ;User defined values for the desired offsets or the values of
        ;j,k, l in x[n-j], x[n-k], x[n-l]
        MOVLF 3, offsetOne
        MOVLF 6, offsetTwo
        MOVLF 9, offsetThree

        ;Length of the array
        MOVLF 8, period         ;length of the array

        rcall  Initial          ;Initialize everything
mainLoop

        rcall updateTerms
        rcall updateAnswer

        bra mainLoop


;;;;;;; TIME SERIES DATA
;
;   The following bytes are stored in program memory.
;   Created by AC
;
;  Choose your Periodic Sequence
;---------------------------------------------------------------
; time series X1
;SimpleTable ; ---> period 2
;db 180,240
;---------------------------------------------------------------
; time series X2
;SimpleTable ; ---> period 4
;db 180,240,200,244
;---------------------------------------------------------------
; time series X3
;SimpleTable ; ---> period 6
;db 180,240,200,244,216,236
;---------------------------------------------------------------
; time series X4
SimpleTable ; ---> period 8
db 180,240,200,244,216,236,160,176
; ---------------------------------------------------------------
```

Fig. 6. Code for the mainline program

TABLE I
VALUES OF $y[n]$ WHEN $x[n]$ IS THE TABLE WITH A PERIOD OF TWO..

$x[n]$ @ **period = 2**	$y[n]$ @ **k = {1,2,3}**	$y[n]$ @ **k = {2,4,6}**	$y[n]$ @ **k = {3,6,9}**	$y[n]$ @ **k = {1,3,5}**
180	45	45	45	45
240	105	60	60	105
180	150	90	45	105
240	210	120	105	150
180	210	135	105	165
240	210	180	105	195
180	210	180	150	225
240	210	240	165	195
180	210	180	150	225
240	210	240	210	195
180	210	180	210	225
240	210	240	210	195
180	210	180	210	225
240	210	240	210	195
180	210	180	210	225
240	210	240	210	195
180	210	180	210	225
240	210	240	210	195
180	210	180	210	225
240	210	240	210	195
180	210	180	210	225
240	210	240	210	195
180	210	180	210	225
240	210	240	210	195
180	210	180	210	225
240	210	240	210	195
180	210	180	210	225
240	210	240	210	195
180	210	180	210	225
240	210	240	210	195
180	210	180	210	225
240	210	240	210	195
180	210	180	210	225
240	210	240	210	195
180	210	180	210	225
240	210	240	210	195
180	210	180	210	225
240	210	240	210	195
180	210	180	210	225
240	210	240	210	195

TABLE II
VALUES OF $y[n]$ WHEN $x[n]$ IS THE TABLE WITH A PERIOD OF FOUR..

$x[n]$ @ **period = 4**	$y[n]$ @ **k = {1,2,3}**	$y[n]$ @ **k = {2,4,6}**	$y[n]$ @ **k = {3,6,9}**	$y[n]$ @ **k = {1,3,5}**
180	45	45	45	45
200	95	50	50	95
240	155	105	60	110
244	216	111	106	166
180	216	150	95	156
200	216	161	110	200
240	216	210	166	221
244	216	222	156	226
180	216	210	155	217
200	216	222	216	200
240	216	210	216	221
244	216	222	216	226
180	216	210	216	217
200	216	222	216	200
240	216	210	216	221
244	216	222	216	226
180	216	210	216	217
200	216	222	216	200
240	216	210	216	221
244	216	222	216	226
180	216	210	216	217
200	216	222	216	200
240	216	210	216	221
244	216	222	216	226
180	216	210	216	217
200	216	222	216	200
240	216	210	216	221
244	216	222	216	226
180	216	210	216	217
200	216	222	216	200
240	216	210	216	221
244	216	222	216	226
180	216	210	216	217
200	216	222	216	200
240	216	210	216	221
244	216	222	216	226
180	216	210	216	217
200	216	222	216	200
240	216	210	216	221
244	216	222	216	226

TABLE III
VALUES OF $y[n]$ WHEN $x[n]$ IS THE TABLE WITH A PERIOD OF SIX..

$x[n]$ @ **period = 6**	$y[n]$ @ **k = {1,2,3}**	$y[n]$ @ **k = {2,4,6}**	$y[n]$ @ **k = {3,6,9}**	$y[n]$ @ **k = {1,3,5}**
180	45	45	45	45
240	105	60	60	105
200	155	95	50	110
244	216	121	106	156
216	225	149	114	175
236	224	180	109	208
180	219	194	151	225
240	218	240	174	209
200	214	199	159	230
244	216	241	212	210
216	225	203	228	234
236	224	239	218	208
180	219	194	212	225
240	218	240	228	209
200	214	199	218	230
244	216	241	212	210
216	225	203	228	234
236	224	239	218	208
180	219	194	212	225
240	218	240	228	209
200	214	199	218	230
244	216	241	212	210
216	225	203	228	234
236	224	239	218	208
180	219	194	212	225
240	218	240	228	209
200	214	199	218	230
244	216	241	212	210
216	225	203	228	234
236	224	239	218	208
180	219	194	212	225
240	218	240	228	209
200	214	199	218	230
244	216	241	212	210
216	225	203	228	234
236	224	239	218	208
180	219	194	212	225
240	218	240	228	209
200	214	199	218	230
244	216	241	212	210

TABLE IV
VALUES OF $y[n]$ WHEN $x[n]$ IS THE TABLE WITH A PERIOD OF EIGHT.

$x[n]$ @ **period = 8**	$y[n]$ @ **k = {1,2,3}**	$y[n]$ @ **k = {2,4,6}**	$y[n]$ @ **k = {3,6,9}**	$y[n]$ @ **k = {1,3,5}**
180	45	45	45	45
240	105	60	60	105
200	155	95	50	110
244	216	121	106	156
216	225	149	114	175
236	224	180	109	208
160	214	189	146	220
176	197	224	158	188
180	188	189	154	209
240	189	224	206	199
200	199	189	208	213
244	216	224	215	196
216	225	189	215	219
236	224	224	207	208
160	214	189	205	220
176	197	224	198	188
180	188	189	198	209
240	189	224	206	199
200	199	189	208	213
244	216	224	215	196
216	225	189	215	219
236	224	224	207	208
160	214	189	205	220
176	197	224	198	188
180	188	189	198	209
240	189	224	206	199
200	199	189	208	213
244	216	224	215	196
216	225	189	215	219
236	224	224	207	208
160	214	189	205	220
176	197	224	198	188
180	188	189	198	209
240	189	224	206	199
200	199	189	208	213
244	216	224	215	196
216	225	189	215	219
236	224	224	207	208
160	214	189	205	220
176	197	224	198	188