HCI

구현 완성도 개선

제출일 : 2018-05-27

담당교수	이강훈 교수님	학 번	2013726091
학 과	컴퓨터소프트웨어학과	이 름	양지용

KWANGWOON UNIVERSITY

● 개요

▶ 기능적 요구조건에 대한 구현완성도

사용자 정의 함수		중첩 지원		표시 항목		그래프 표시 영역	
f(x)	O	색상	O	원점	O	원점위치	O
g(x)	O	라벨	O	좌표축	O	정의역 구간	O
h(x)	O			눈금	O		
				교점	O		

▶ 오픈소스 라이브러리 의존성

▶ math.js

▶ https://plot.ly/

• math.js에서 지원해주는 그래프 Tool

▶ http://www.blueb.co.kr/?c=2/31&cat=%EA%B8%B0%ED%83%80%EB%A9%94%EB%89%B4&uid=3969

• 메뉴 원형 펼치기 위한 JavaScript 참조

▶ http://itzone.tistory.com/272

• textarea에서 커서 위치 파악하기 위한 참조

▶ 사용성 향상에 기여하는 핵심적인 상호작용 방식

• 함수 그래프를 그리기 위하여 노란 버튼을 하나 더 추가하였습니다.

• 그래프를 확인하기 위하여 전체적인 UI 크기를 변경 하였으며, 각 버튼간의 거리에 여유를 두었습니다.

● 본문

▶ 향후 개선 계획의 이행 사항

▶ 삼각함수 이용시 degree만을 사용하는 것이 아니라 라디안을 사용하는 방식도 추가 하여 사용자가 degree와 radian간의 교환을 하지 않더라도 수행이 가능하도록 할 예정입니다.

• Radian으로의 각도 사용은 넣지 않았습니다.

▶ Del버튼 클릭시 간혹 커서를 이용하여 다시 삭제할 부분을 선택해주어야 하는 불편함이 있습니다. 현재 커서 위치, 수정위치를 기억하고 있도록 수정 하겠습니다.

• 현재 커서 위치를 키의 경우의 수를 더 생각하여 오류 발생률을 이전보다 줄였습니다.

▶ 계산 결과를 도출하는 = 키를 누를시 value를 0으로 수정하는 것이 아닌 현재 결과값으로 개선하고, 이어서 연산 키를 누를 시 연산이 가능하도록 개선 하겠습니다.

• 연산키를 누를 시 이어서 연산이 가능하며, 숫자를 누를 경우 새로운 계산이 시작되도록 변경 하였습니다.

▶ 반올림의 경우 사용자가 표시할 소수점을 선택할 수 잇는 부분을 추가하여 깔끔한 결과 도출이 가능하다면 개선하도록 하겠습니다.

• 무조건 소수점 3번째 자리 까지 나타내는 반올림으로 수정하였습니다.

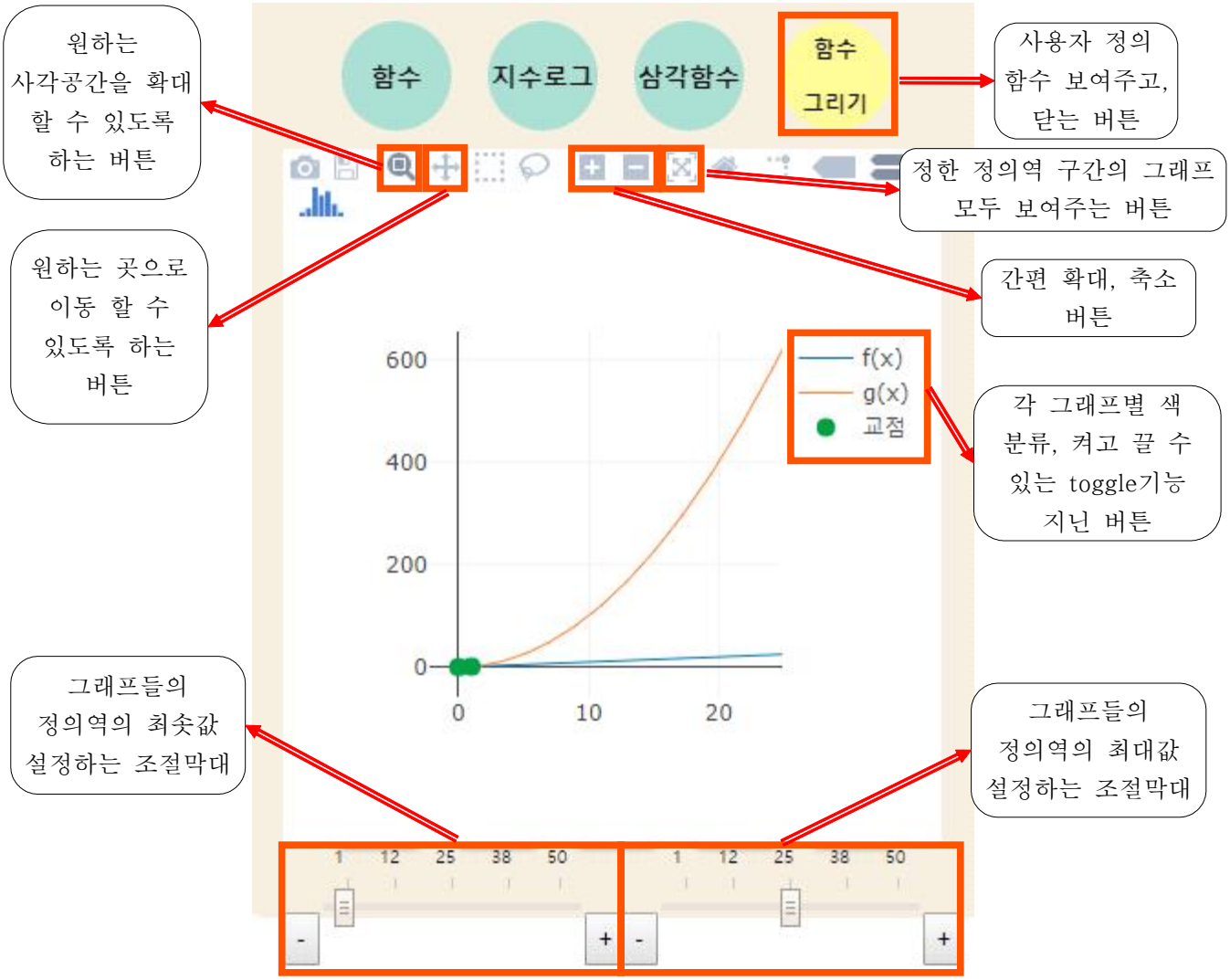
▶ 자세한 사용방법이 없을 경우 사용자가 math.js에 맞게 사용을 하는데에 힘들다는 점이 있습니다. 이는 자세한 설명을 통해 사용자가 사용가능하도록 할 예정입니다.

• 4번째 과제를 진행 할 때 사용 설명을 위한 문서를 만들어 전달 할 예정입니다.

▶ 0을 가장 먼저 입력할 경우 스크립트 상 0을 인식하지 않아 0이 표시되지 않는 점은 개선이 필요한 사항이라고 생각합니다.

• 첫 입력인지 확인하는 boolean 변수를 통해 개선하였습니다. 이제 0.xxx형식의 소수점을 먼저 적어도 무방합니다.

▶ 그래프 Interface사용방법



▶ 개선 부분에 대한 세부 구현 방법

- 계산 결과 도출 이후 이어 연산이 가능하도록 구현
- 0을 먼저 입력하여도 입력이 되도록 구현

```
//첫 입력으로 0이 입력 될때, 연속 계산을 할 때
// 인식 위한 check boolean변수
var bool_click = false;
$(document).ready(function(){
    var parser = math.parser();
    var exAns = '';       //이전값 기억 변수
    var eexAns = '';      //AC 이전값 기억 변수
    //displayValue는 공백으로, 결과창은 0으로 표시 위함.
    $('#result').text('0');

    $('.key').each(function(index, key){
        $(this).click(function(e){
            if(bool_click == false){
                    $('#result').text(displayValue);
            }
            //계산완료 버튼을 눌렀을 때 화면과 value값을 지워준다.
            //연속 계산위해 연산자 클릭을 할 경우 해당 if문 내에서 exAns를 이용하여
            //계산 결과를 도출 해준다.
            if(displayValue == '0' && bool_click == false){
                displayValue = '';
                $('#result').text(displayValue);
            }
```

▶ bool_click 변수를 통해 첫 입력인지 파악을 하고 displayValue와 결과창의 text를 변경해준다.

▶ 이를 통해 첫 입력이 0일 경우 입력받아 저장하게 된다.

▶ 계산 완료 버튼, AC버튼을 제외한 나머지 버튼을 클릭할 경우 bool_click=true로 변경한다.

```
        else if(okey.text() == '+' || okey.text() == '*' || okey.text() == '-' ||
            okey.text() == '/' || okey.text() == '%'){
                var text;
                text = okey.text();
                if(bool_click == false){
                    $('#result').text(eexAns);
                    currentPos = eexAns.length;
                }
                else{
                    $('#result').text(displayValue);
                    currentPos = displayValue.length;
                }
                currentPos = insertAtCursor("result", text, currentPos, text.length);
                bool_click = true;
        }
```

▶ 연산버튼을 눌렀을 경우 결과가 완료 되었는지 확인하고

▶ AC를 눌러 값이 초기화 된 경우 이미 저장 되어있는 값이 나오지 않기 위해 exAns가 아닌 eexAns를 결과창에 삽입하였습니다.

▶ currentPos도 넣어주는 값에 따라 변화 시켜 커서 위치를 정상적으로 만들어 주었습니다.

• 사용자 정의 함수 그래프로 그려주기

```
                    //한번 계산해서 function인지 파악, 아니라면 round씌워서 재계산
                exdisplayvalue = displayValue;
                var tempResult = displayValue = parser.eval(displayValue).toString();
                var tokens = tempResult.split(' ');
                var funcTokens;
                if(tokens[0] == 'function')
                {
                        displayValue = tokens[0];
                        functokens = exdisplayvalue.split('=');
                        dict[functokens[0]] = functokens[1];
                        dataSize++;
                        $('#result').text(displayValue);
                        displayValue = '0';
                        exAns = displayValue;
                        eexAns = displayValue;
                        currentPos = 0;
                }
```

▶ 사용자 정의 함수 생성을 할 경우 전역 변수인 dict에 해당 함수들을 모두 저장합니다.

▶ 사용자 정의 함수 개수인 dataSize도 저장 하여 둡니다.

```
function draw(dict, dataSize) {
  try {
    var expression;
    var expr;
    var variable;
    var dataCount=0;
    var trace = {};
    //여러 그래프 정보들을 담아갈 객체
    var mainObject = new Array();

    for(var key in dict){
        expression = dict[key];
        //사용자 정의 함수들에서 식을 가져와 해당 expression을 그래프에 맞추기 위해 작업
        var expr = math.compile(expression);

        //key(f(x))에서 x를 잘라 x인지 다른 문자인지 판단하여 밑의 expr.eval부분에 삽입
        //x만 그래프로 그려줄 수 있음.
        variable = key.substr(2,1);
        // evaluate the expression repeatedly for different values of x
        if(variable == 'x'){
                var xValues = math.range(parseInt(drawMin), parseInt(drawMax), 0.1).toArray();
                var yValues = xValues.map(function (x) {
                    return expr.eval({x: x});
                });
                trace[dataCount] = {
                        type : 'scatter',
                        x:xValues,
                        y:yValues,
                        mode:'lines',
                        name: key,
                        line:{
                            width: 1
                        }
                }
        mainObject[dataCount] = trace[dataCount];
        dataCount++;
        }
    }
```

▶ 그래프를 그려줄 때에 사용자 정의 함수의 정의역을 담당하는 변수가 x인지 파악해주고, x라면 그려줄 Object를 저장하고있는 배열 mainObject에 trace라는 json형식의 정보를 저장해 둡니다.

▶ 각 그래프들의 형식은 정해져 있으며, 색의 경우 plot에서 자동으로 구별이 가도록 지정해 주어 따로 설정하지 않았습니다.

```
411                     var layout = {
412                             width : 360,
413                             height: 380,
414                     };
415                     Plotly.newPlot('plot', mainObject, layout);
```

▶ 마지막으로 그래프를 그려줄 창 크기를 지정하고 저장한 mainObject를 그려줌으로써 그래프를 그리게 됩니다.

• 특수 점 교점 나타내기

```
353             var tempdt = dataCount;
354             var flag = 0;
355             //교점 그려주기
356             for(var key1 in dict){
357                     for(var key2 in dict){
358                             //x인 함수만 골라내기 위한 if문
359                             if(key1.substr(2,1) != 'x' || key2.substr(2,1) != 'x'){
360                                     continue;
361                             }
362                             if(flag == 0){
363                                     //중복 방지 위한 if문, flag
364                                     //못 만났으면 그냥 넘어가고 만났으면 flag=1설정 이후 넘어가기
365                                     if(key1 == key2){
366                                             flag=1;
367                                             continue;
368                                     }
369                                     else{
370                                         continue;
371                                     }
372                             }
373                             //교점 구하기 위해 식 수정 f(x)-g(x)=0이 되는 점 찾기
374                             var temp = "-"+"("+dict[key2]+")";
375                             var texp = math.compile(dict[key1]+temp);
376                             var realexp = math.compile(dict[key1]);
377
378                             var shareP = math.range(parseInt(drawMin), parseInt(drawMax),0.1).toArray();
379                             var xValues = shareP.map(function(x){
380                                     //정확한 0을 계산하지 못하므로 근사치를 계산하여 교점으로 판단
381                                     //eval하면 x좌표가 튀어나옴
382                                         if(texp.eval({x: x}) >= -0.00001 && texp.eval({x: x}) <= 0.00001)
383                                         {
384                                                 return x;
385                                         }
386                             });
387                             //해당 x좌표로 y좌표 매핑
388                             var yValues = shareP.map(function(x){
389                                 return realexp.eval({x:x});
390                             });
```

▶ 함수 정보를 담아둔 dict를 이중for문을 통해 돌면서 서로의 교점을 확인하는 작업을 합니다.

▶ 중복으로 교점을 구하는 것을 방지하기 위해 flag를 두었습니다.

▶ 교점의 경우 f(x)−g(x)=0이 되는 점을 f(x)에 대입하여 y좌표를 구하는 방식으로 진행하였습니다.

▶ 그러나 f(x)−g(x)=0이 되는 점을 계산하지 못하는 방식이라 근사치를 이용하여 교점을 판단하였습니다. x좌표의 간격이 0.1이 아닌 작으면 작을수록 교점 계산에 탁월 할 것입니다.

```
392                         trace[dataCount] = {
393                                 x:xValues,
394                                 y:yValues,
395                                 mode:'markers',
396                                 name:'교점',
397                                 marker:{
398                                         size : 10
399                                 }
400                         }
401                         mainObject[dataCount] = trace[dataCount];
402                         dataCount++;
```

▶ 교점의 경우도 json 형태로 저장하여 mainObject에 더하여 구분할 수 있도록 하였습니다.

- 정의역 설정

```
530                     //정의역으로 사용할 min값, max값을 비교하여
531                     //min을 클릭하였을 때 max를 넘지 않게
532                     //max를 클릭하였을 때 min보다 작지 않게 조정해준다.
533                 if(p == getObject("slidecontainer-max")){
534                         o = getObject("rmax");
535                         dtlist = getObject("tickmarks-max");
536                         lb = getObject("lb_range_max");
537                         if(r < drawMin){
538                                 r = drawMax;
539                         }
540                         else{
541                                 drawMax = r;
542                         }
543                 }
544                 else if(p == getObject("slidecontainer-min")){
545                         o = getObject("rmin");
546                         dtlist = getObject("tickmarks-min");
547                         lb = getObject("lb_range_min");
548                         if(drawMax < r){
549                                 r = drawMin;
550                         }
551                         else{
552                                 drawMin = r;
553                         }
554                 }
555             o.min = parseInt(r) - 25;
556             o.max = parseInt(r) + 25;
557
558             dtlist.options[0].value = o.min;
559             dtlist.options[1].value = parseInt(r) -13;
560             dtlist.options[2].value = parseInt(r);
561             dtlist.options[3].value = parseInt(r) + 13;
562             dtlist.options[4].value = o.max;
```

▶ min, max 슬라이더에 따라 정의역을 설정해준다. min과 max간의 값 침범이 없기 위한 if~else문을 추가하였다.

```
569         function changetick(v, p){
570             var o;
571             if(p == getObject("slidecontainer-max")){
572                     o = getObject("rmax");
573             }
574             else if(p == getObject("slidecontainer-min")){
575                     o = getObject("rmin");
576             }
577             if(v == "-"){
578                     range(parseInt(--o.value), p);
579             }
580             else if(v == "+"){
581                     range(parseInt(++o.value), p);
582             }
583         }
```

▶ 각 슬라이더 양 옆에 달려있는 버튼을 클릭할 경우 처리해주는 함수입니다.

• 반올림 하여 표현

```
239                 else{
240                         displayValue = parser.eval('round(' + displayValue + ', 3)').toString();
241                         $('#result').text(displayValue);
242                         exAns = displayValue;
243                         eexAns = displayValue;
244                         //반올림 해주면서 currentPos값이 변하여 결과값 도출 이후에
245                         //currentPos값을 맨 뒤로 보내기 위함.
246                         currentPos = displayValue.length;
247                         displayValue = '0';
248                 }
```

▶ 계산 완료 버튼을 누를 경우 수식을 round 함수를 이용하여 소수점 세 번째 자리까지 나타내도록 하였습니다.

▶ 실제 문제에 대한 사용 예시

-> 3개 이상 공학 계산 문제에 대한 해결 과정 (캡처 이미지를 이용한 스토리보드 형식)

-> YouTube 시연 동영상 링크 (3분 이내)

- f(x)=x^2저장, g(x)=x저장, h(y)=y+1
- 1부터 5까지 연속으로 더한 후 * 10
- f(x)=sin(x)+1 지정 이후 f()를 이용해서 f(pi)값 계산

문제 풀이 위한 계산기 사용 영상 링크 : https://youtu.be/2Z5aLq2aO-E

● 논의

▶ 구현 측면에서 성공적인 부분과 실패한 부분

- 우연히 math.js에서 지원하게되는 plot.ly.js를 찾게 되어 쉽게 그래프를 구현하였으며, 사용자 정의 함수를 통해 plot의 규격에 맞게 바꾸어주는 작업만 할 수 있어 작업의 양이 급격히 감소하였습니다.
- 이전에 개선 계획으로 올렸던 것 중 삼각함수 이용시 radian과 degree간의 교환이 없도록 하는 것을 제외하고 모두 개선하여 성공적인 구현이라 생각합니다.

▶ 사용성 측면에서 긍정적인 측면과 부정적인 측면

▶ 긍정적인 측면

- 나온 계산 결과값으로 바로 계산을 가능하게 수정하여 번거러움을 줄였습니다.
- 0.xxx 의 계산이 되지 않았던 점을 수정하여 사용범위를 올렸습니다.

▶ 부정적인 측면

- 현재 커서위치를 저장하고 수정하는데에 경우의 수가 다수 존재하여 다시 클릭 해주어야 하는 경우가 아직 존재합니다.

▶ 과제 #3에 대한 전반적인 자체 평가 및 향후 개선 계획

- 삼각함수 이용시 degree만을 사용하는 것이 아니라 라디안을 사용하는 방식도 추가 하여 사용자가 degree와 radian간의 교환을 하지 않더라도 수행이 가능하도록 할 예정입니다.
- 그래프간의 교점을 구하는데에 있어 x좌표의 간격이 존재하여 정확하게 교점이 나오지 않는 경우가 있습니다. 함수간의 값을 비교하여 교점의 유무만을 파악하여 존재한다는 것을 알리기 위한 점을 찍는 것으로 변경을 할 예정입니다.
- 그래프 그릴 경우 약간 튀어나오는 +,- 버튼을 전체적인 UI 크기를 다시 조절하여 계산기 안에 들어가도록 할 예정입니다.