Кобак Федор

функция для построения лагранжа:

```
function  c = lagIntPoly(xVec , y , g)

   % check is interpolation point correct
   if(~isIntDataCorrect(xVec , y))
      disp('data not correct');
      return
   end

   counter = 1;
   syms x;

   c = 0;

   for i = xVec

      tempDen = 1; % temp denominator
      tempNum = 1; % temp numerator

      for j = xVec
         if(i ~= j)
            tempDen = tempDen * (i - j);
            tempNum = tempNum*(x - j);
         end
      end

      %disp(['denaminator number ', num2str(counter) , ' is ' , num2str(tempDen)]);
      %disp(['numerator number ', num2str(counter) , ' is ' , char(tempNum)]);
      c = c + (tempNum/tempDen)  * y(counter);
      counter = counter + 1;
   end

   % chart
   if g == 1

       resultY =  subs(c , xVec);

       subplot(1 , 2 ,1);
       plot(xVec, y);
       title('input data');

       subplot(1 , 2 ,2);
       plot(xVec , resultY);
       title('interpolation data');

   end
```

```
end
```

функция для построения ньютона

```
%function for building lagrange Interpolation
%by Fedor Kobak github/Dranikf
% y - array of values of function
% point - a point for which interpolation
% iType - is a type of interpolation {
%       0 - if its need to use point parameter
%       1 - if its need to build first interpolation poly (point is ignore)
%       2 - if its need to build second interpolation poly (point is ignore)
%       }
%       dispT - is need to show temp data
function [polyn , result] = NewtonIntPoly(x , y , a , b , iType, dispT , point)

        if(a > b)
                disp('a must be smaller then b');
                return;
        end

        region = b - a;
        h = region / (numel(y) - 1); % step

        x = a:h:b;
        step = 1; % this is step in indexes

        if(~iType)
                if(point > b)
                        disp('point is not in regon');
                        return;
                end

                [step, PIndex] = getTypeOfNew(x , point)

                if(dispT)
                        if(step == 1)
                                disp('need to use first interpolation poly');
                        else
                                disp('need to use second interpolation poly');
                        end

                        disp(['start x value is '  num2str(x(PIndex))]);

                end


        elseif(iType == 1)
                PIndex = 1;
```

```
        step = 1;
elseif(iType == 2)
        PIndex = numel(y);
        step = -1;
else
        disp('uncnown type');
        return;
end


q = sym('x');
q = (q - x(PIndex)) / h;
if(step == 1)
        endInd = numel(x);
else
        endInd = 1;
end

indexesVector = PIndex:step:endInd;

if(step == 1)
        endDiffs = finDiffTable(x(indexesVector) , y(indexesVector));
        diffIndex = 1;
else
        % in case of indexesVector builded for second formula, we need to inverce it for
        % building table of differences
        inverceInVec = indexesVector(numel(indexesVector):-1:1);
        endDiffs = finDiffTable(x(inverceInVec) , y(inverceInVec));
        diffIndex = numel(indexesVector) - 1;
end

tableSize = size(endDiffs);
endDiffs = endDiffs(: , 3:tableSize(2)); %  cut a bit diffsTable
polyn = sym(y(indexesVector(1)));
qProd = q;

if (dispT)

        disp('q is ' )
        q

        disp('end diffs table is' );

        endDiffs

end

for i = 2:numel(indexesVector)
```

```
            if (dispT)
                disp(['poly part number ' , num2str(i) ,' is ' ]);
                diffIndex
                (qProd * endDiffs(diffIndex, i-1)) / factorial(i -1)
            end

            polyn = polyn + (qProd*endDiffs(diffIndex, i-1)) / factorial(i - 1);
            qProd = qProd*(q - (i-1)*step); % q(q - 1)(q - 2)... or q(q + 1)(q + 2)...

            if diffIndex ~= 1
                diffIndex = diffIndex - 1;
            end
        end

        if (~iType)
            result = subs(polyn , point);
        end

end
```

Сценарий выполнения для данного варианта:

```
x = [0.847 1.546 1.834 2.647 2.91];
y = [-1.104 1.042 0.029 -0.344 -0.449]

disp('Lagrange interpolation poly');
lagPoly = lagIntPoly(x , y, 0)
disp('first Newton intrtpolation poly');
nPoly = NewtonIntPolyArr(x , y, x(1) , 2.91 , 1 , false , 0)

disp('L(x1 + x2)')
subs(lagPoly , x(1) + x(2));
disp('N(x1+x2)')
subs(nPoly , x(1) + x(2));

Y2 = subs(lagPoly , x);
Y3 = subs(nPoly , x);

plot(x, [Y2 ; Y3]);

legend('lagrange' , 'newton')
```

Результат

```
>> lab1resultScript

y =

   -1.1040    1.0420    0.0290   -0.3440   -0.4490

Lagrange interpolation poly

lagPoly =

(774619135907725312*(x - 291/100)*(x - 773/500)*(x - 917/500)*(x - 847/1000))/954192276689337625 - (310748374288564224*(x - 291/100)*(x - 773/500)*(x - 917/500)*(x - 2647/1000))/721117198238440625 + (130604389193744384*(x - 291/100)*(x - 773/500)*(x - 847/1000)*(x - 2647/1000))/1119881916572375375 - (23463754058600284 16*(x - 291/100)*(x - 917/500)*(x - 847/1000)*(x - 2647/1000))/680771132606606375 - (505529058172338176*(x - 773/500)*(x - 917/500)*(x - 847/1000)*(x - 2647/1000))/896564216325879875

first Newton intrtpolation poly

nPoly =

(8584*x)/2063 - (3159*((4000*x)/2063 - 3388/2063)*((4000*x)/2063 - 5451/2063))/2000 + (3799*((4000*x)/2063 - 3388/2063)*((4000*x)/2063 - 5451/2063)*((4000*x)/2063 - 7514/2063))/6000 - (4171*((4000*x)/2063 - 3388/2063)*((4000*x)/2063 - 5451/2063)*((4000*x)/2063 - 7514/2063)*((4000*x)/2063 - 9577/2063))/24000 - 47741/10315

L(x1 + x2)
N(x1+x2)
```

ГРАФИК