МІНІСТЕРСТВО ОСВІТИ ІНАУКИ УКРАЇНИ
НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
„КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
імені ІГОРЯ СІКОРСЬКОГО”

ЧИСЛОВІ МЕТОДИ
КОНСПЕКТ ЛЕКЦІЙ

Рекомендовано Методичною радою КПІ ім. Ігоря Сікорського
як навчальний посібник для студентів,
які навчаються за спеціальністю 151 «Автоматизація та комп’ютерно-інтегровані технології»

Київ
КПІ ім.. Ігоря Сікорського
2019

Числові методи: конспект лекцій [Електронний ресурс] : навч. посіб. для студентів спеціальності 151 «Автоматизація та комп’ютерно-інтегровані технології» / КПІ ім.. Ігоря Сікорського; уклад.: О. В. Ситніков. – Електронні текстові дані (1 файл: 3,36Мбайт). – Київ: КПІ ім.Ігоря Сікорського, 2019. – 165с.

Гриф надано Методичною радою КПІ ім. Ігоря Сікорського (протокол № від 25 .04.2019р.)
за поданням Вченої ради Інженерно-хімічного факультету (протокол № від 28 .01.2019 р.)

Електронне мережне навчальне видання

ЧИСЛОВІ МЕТОДИ

Конспект лекцій

Укладачі: *Ситніков Олексій Володимирович*, ст.викл.

Відповідальний редактор *Жученко А.І.*, д.т.н., проф.

Рецензент *Степанюк А.Р.*, к.т.н., доц., доц., НТУУ «КПІ ім.Ігоря Сікорського»

Зміст

Розділ 1. Операції над поліномами.

Лекція 1. Обчислення значення полінома за схемою Хорнера при дійсному та комплексному значенні аргументу

Запишемо поліном *A(x)* у такому вигляді

$$A(x) = a_n x^n + a_{n-1} x^{n-1} + ... + a_1 x + a_{0.}.$$

Цей же поліном можна представити і так

$$A(x) = (...((a_n x + a_{n-1})x + a_{n-2})x + ... + a_1)x + a_0 \ .$$

Розкриваючи дужки в легко пересвідчитись в тому, що та ідентичні. Формула представляє так звану схему Хорнера. Послідовність обчислень від самих внутрішніх дужок послідовно до самих зовнішніх (природня послідовність при обчисленні арифметичних виразів). Програмна реалізація схеми Хорнера – підпрограма-функція HorReal.

```
function HorReal(A:Coef: x:real):real;
        var s,n :integer;
            r : real;
        begin
              n:=round(A[-1]); r:=A[n];
                for s:= n-1   downto   0  do
                      r:=r*x+A[s];
            HorReal:=r
         end;
```

Математики стверджують, що схема Хорнера – практично найкраща(найекономніша) з можливих для обчислення значення полінома- з ними можна погодитись. В подальшому ми будемо віддавати їй перевагу.

Скористаемось тією ж таки схемою Хорнера щоб обчислити значення полінома при комплексному значенні *x=R+jI*, де *R,I* – дійсна та уявна частина *x, j*= $\sqrt{-1}$ – уявна одиниця.

Один крок (виток) в циклі, що реалізує схему Хорнера при комплексному x, можна представити так

$$Re^H + jIm^H = (Re^c + jIm^c)(R + jI) + a_s,$$

де, як уже згадувалось, *R+jI* – значення *x*, *Re+jIm* – поточне (в процесі реалізації схеми Хорнера) значення полінома, a_s – черговий коєфіцієнт полінома.

Припустимо, що при комплексному *x* і значення полінома має бути в загальному випадку комплексним *Re+jIm*.

$Re^c + jIm^c$ –„старе” (отримане на попередньому кроці) значення полінома.
$Re^H + jIm^H$ –„нове” (оновлене) його значення. Комплексну рівність можна представити двома дійсними рівностями.

$$Re^H = Re^c \cdot R - Im^c \cdot I + a_s,$$
$$m^H = Re^c \cdot I + Im^c \cdot R.$$

Співвідношення покладено в основу підпрограми *HorComp*.

```
procedure HorComp (A:Coef; R,I:real; var Re, Im:real);
    var s,n:integer;
  begin
      n:=round(A[-1]); Re:=A[n]; Im:=0;
      for  s:=n-1 downto  0  do
          begin
              R1:=Re*R-Im*I+A[s];
              Im:=Re*I+Im*R;
              Re:=R1
          end
  end;
```

Верхні індекси(‘н’ та ‘с’-нове, старе) в підпрограмі *HorComp* не фігурують, оскільки ми користуємося тією властивістю оператора присвоювання, що в його правій частині використовується „старі”, а в лівій „нові” значення ідентифікаторів, що повторюються. От тільки довелось ввести нову (проміжну) змінну R1:*real*, щоб запам’ятовувати оновлене значення *Re* не втрачаючи при цьому його старе значення, необхідне для оновлення *Im*. Після того, як необхідність у зберіганні старого *Re* відпадає, йому повертається його значення, що тимчасово зберігалось в R1.

Багато прикладних проблем, наприклад, інтерполяція, згладжування, розв’язання диференціальних рівнянь та їх систем, формування ряду динамічних характеристик, як-то перехідних, імпульсних, частотних і т.п. – мають розв’язок у вигляді функціональних залежностей (явних чи неявних), сприйняття яких значно полегшується, якщо їх представити графічно – у вигляді графіків, годографів, систем ізоліній, аксонометричних зображень поверхонь. Для цього доцільно розглянути наявні і, можливо, створити додаткові програмні засоби, які б перекривали окреслене коло проблем. Нижче пропонується елементарні алгоритми по формуванню системи координат під графіки та(чи) годографи.

Розглянемо проблему формування на екрані монітора системи координат. На прямокутну ділянку розміром *L* пікселів по горизонталі та *H* пікселів по вертикалі відображується прямокутна область площини

XOY, обмежена по горизонталі значеннями *Xmin, Xmax*, а по вертикалі, відповідно, *Ymin* та *Ymax*.

Можливі два варіанти постановки задачі: 1) шкали на горизонтальній та вертикальній осях не узгоджуються між собою, що характерно для формування графіків, і 2) шкали узгоджуються так, що ціна відрізка довжиною в 1 піксель по горизонталі відноситься до ціни відрізка в 1 піксель по вертикалі, як параметри *Aw* і *Bw* , що повертаються процедурою *GetAspectRatio(Aw, Bw)*. Другий варіант характерний для формування годографів. Зокрема, у випадку, коли по горизонтальній та вертикальній осях відкладаються величини однієї і тієї ж фізичної природи (однієї розмірності) і важливо зберегти природну форму годографа (наприклад, коло має бути відтворено саме як коло, а не як еліпс).

Домовимось позначати коефіцієнтом Hj відношення Bw/Aw, тобто *Hj=Bw/Aw*.

Вибір першого чи другого з розглянутих варіантів будемо реалізувати через параметр *God:boolean* в процедурі *X0Y0*, заголовок якої матиме вигляд

```
Procedure X0Y0 (God:boolean);
```

Тут *God=true*, якщо система координат формується для годографа і *God=false* в іншому випадку.

Визначення параметрів системи координат починаємо з обчислення *Dx* - ціни одного пікселя по горизонталі та *Dy*- по вертикалі, за умови, що ширина *L* та висота *H* заданої області на екрані використовуються повністю (це характерно для *God=false*).

$$Dx=(Xmax-Xmin)/L;$$
$$Dy=(Ymax-Ymin)/H.$$

Отримані таким чином значення *Dx* та *Dy* при *God=true* перевіряються на виконання умови *Dy/Dx=Hj*. Коли, наприклад, $Dy>Dx\cdot Hj$, то приймається рішення перерахувати *Dx* і, відповідно, ширину прямокутника (*L0* замість *L*), в протилежному випадку перераховується *Dy* (*H0* замість *H*).

Після визначення фактичних розмірів прямокутника *L0* x *H0*, він розміщується в центрі екрана (координати *Xu, Yu* визначають лівий верхній кут прямокутника). Розраховуються також *X0, Y0* – екранні координати точки, куди відображується початок системи координат *X0Y*. Для подальшої розмітки шкал, нанесення координатної сітки та оцифровування шкал визначаються початкові значення *Xn, Xk, Yn, Yk*.

```
procedure X0Y0(God:boolean);
begin
  Dx:=(Xmax-Xmin)/L; L0:=L;
```

```
  Dy:=(Ymax-Ymin)/H; H0:=H;
  if God then
    if Dy>Dx*Hj
      then
        begin
           L0:=round((Xmax-Xmin)/Dy*Hj);
           if L0<>0 then Dx:=(Xmax-Xmin)/L0
                    else  Dx:=0
        end
      else
        begin
          H0:=round((Ymax-Ymin)/(Dx*Hj));
          if H0<>0
            then Dy:=(Ymax-Ymin)/H0
            else  Dy:=0
        end;
    Xu:=(GetMaxX-L0) div 2; Yu:=(GetMaxY-H0) div 2;
    X0:=Xu-round(Xmin/Dx); Y0:=Yu+round(Ymax/Dy);
    Xn:=Xmin; Xk:=Xmax; Yn:=Ymin; Yk:=Ymax
end;
```

Вигляд системи координат, сформованої процедурою X0Y0, представлено на рис.1.

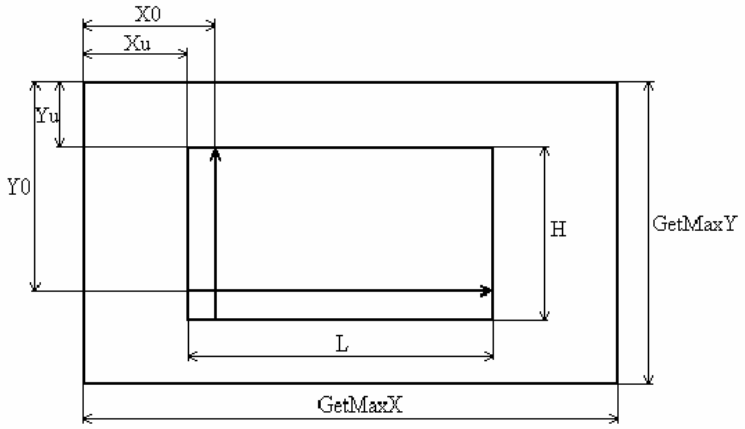

Рис.1.Схема параметрів системи координат, сформованих процедурою *X0Y0* при *God=false*.

Щоб спростити діалог користувача з програмою в процесі її виконання, розроблено ряд елементарних програмних засобів, які зосереджено в модулі *Serv*.

Почнемо з процедури *Clear*, що призначена для стирання зображення в межах прямокутної області, яка задається координатами *X*1,*Y*1 лівого верхнього та *X*2,*Y*2 правого нижнього її кутів (як прийнято у *Turbo Pascal*'і).

```
procedure Clear(X1,Y1,X2,Y2:integer);
begin
  SetViewPort(X1,Y1,X2,Y2,true);
  ClearViewPort;
  SetViewPort(0,0,GetMaxX-1,GetMaxY-1,false)
end;
```

В цій процедурі встановлюється тимчасове графічне вікно відповідного розміру, після стирання зображення в межах якого вікно розширюється до розмірів екрана. Зрозуміло, що можна користуватися цією процедурою лише за умови, що вікном є весь екран, бо інакше поточне вікно після виклику *Clear* буде ліквідовано.
Для ведення діалогу з користувачем домовимось формувати пропозиції (поточні меню та підменю), вводити нові значення параметрів, тощо у верхньому рядку екрана. Для того, щоб стерти зображення, яке знаходиться там в поточний момент, і тим самим звільнити місце для чергового зображення, будемо використовувати процедуру *PutA*.

```
procedure PutA;
begin
  Clear(0,0,GetMaxX-1,10)
end;
```

Для виведення поточного меню у вигляді тексту *Text* в першому рядку екрана призначена процедура *Ou*.

```
procedure Ou(Text:String);
begin
  OutTextXY(0,0,Text)
end;
```

Для введення з клавіатури змінної типу real призначена процедура *Our*, а типу integer - *Oui* відповідно.

```
procedure Our(Text:String; var X:real);
begin
  PutA;
  OutTextXY(150,0,'Введіть    '+Text);
  GotoXY(1,1);
  read(X)
end;

procedure Oui(Text:String; var X:integer);
begin
  PutA;
  OuttextXy(150,0,' Введіть  ' + Text);
```

```
  GotoXY(1,1);
  read(X)
end;
```
Змінна *Text* несе інформацію, що пояснює, яку саме величину треба вводити. Її ехо-повтор виводиться в лівий верхній кут екрана (перед текстом ‘Введіть ’ + *Text*).
При реалізації багатокомпонентних меню зручно інформацію про поточні значення параметрів, що входять до нього, або просто можуть представляти інтерес для користувача в поточний момент, виводити в так званому інформаційному рядку, який розташовано в нижній частині екрана. Домовимось формувати інформаційний рядок в змінній *Ts:String*[80], звільняти місце для неї, як було вказано вище, і виводити *Ts* у відповідному місці процедурою *Info*.

```
procedure Info;
begin
  Clear(0,GetMaxY-10,GetMaxX-1,GetMaxy-1);
  OutTextXY(0,GetMaxY-10,Ts)
end;
```

Наступна процедура, що входить до модуля, – це *X0Y0*. Але вона була описана у попередньому параграфі.
Горизонтальні та вертикальні прямі для зображення відповідних осей системи координат формує процедура *SystCoor*.
```
procedure SystCoor;
begin
  ClearDevice;
  SetWriteMode(1);
  Line(Xu-10,Y0,Xu+L0+10,Y0);
  Line(X0,Yu+H0+10,X0,Yu-10);
  SetWriteMode(0)
end;
```
Процедура передбачає малювання осей інверсним кольором.
Формування на екрані ламаної, що зображує графік функції, заданої масивом *Mo:CoefL*, кольором *C* реалізується процедурою *Graphic* (type *CoefL = array*[-1..601] *of real*).

```
procedure Graphic(Mo:CoefL; C:integer);
var x,y,s:integer;
begin
  if C>=0 then SetColor(C)
          else  SetWriteMode(1);
  MoveTo(Xu,Y0-round(Mo[0]/Dy));
  for s:=1 to L do
```

```
    begin
      x:=X0+s; y:=Y0-round(Mo[s]/Dy);
      LineTo(x,y);
      if  C<0 then PutPixel(x,y,GetMaxColor-GetPixel(x,y))
    end;
  if C>=0 then SetColor(15)
          else  SetWriteMode(0)
end;
```

Тут значенням *C*<0 задається інверсний колір графіка.
Оскільки всі засоби, розглянуті вище, орієнтовані на використання в графічному режимі, було визнано доцільним при підключенні модуля Serv автоматично ініціювати графічний режим за допомогою процедури *InGraph*.

```
procedure InGraph;
var E,Gd,Gm:integer;
    Aw,Bw:word;
begin
  Gd:=Detect;
  InitGraph(Gd,Gm,' ');
  E:=GraphResult;
  if E<>0
    then
      begin
        write(GraphErrorMsg(E));
        readln; Halt(1)
      end
    else
      begin
        GetAspectRatio(Aw,Bw);
        Hj:=Bw/Aw;
        L:=GetMaxX-139;
        H:=GetMaxY-59
       end
end;
```

Основні ідеї та алгоритми формування на екрані монітора графіків функцій було роозглянуто в п.1.1. Продемонструємо їх в роботі програмою Graphics.

```
Program Graphics;
 uses Crt, Graph, Serv, Groms;
 const Nvr:integer=1;
      A:real=-0.5;
      B:real=3;
      K:real=2;
 var Mo:CoefL;
```

```
function F(x:real):real;
  begin
   case Nvr of
    1:F:=sin(x);
    2:F:=cos(k*x);
    3:F:=cos(B*x)+A*sin(k*B*x);
    4:F:=1-exp(A*x)*(cos(B*x)-A/B*sin(B*x))
   end
  end;
procedure Coefs;
  var T1:string;
      J1:char;
  begin
    repeat
      PutA; Ou('0-exit, 1-A, 2-B, 3-k');
      Str(A:1:3, T10); Ts:='A='+T10;
      Str(B:1:3, T10); Ts:=Ts+', B='+T10;
      Str(k:1:3, T10); Ts:=Ts+', k='+T10;
      case Nvr of
        1:T1:='sin(x)';
        2:T1:='cos(k*x)';
        3:T1:='cos(B*x)+A*sin(k*B*x)';
        4:T1:='1-exp(A*x)*(cos(B*x)-A/B*sin(B*x))'
      end;
      Ts:= Ts+'   (F='+T1+')';
      Info; J1:=ReadKey;
      case J1 of
       '1':Our('A',A);
       '2':Our('B',B);
       '3':Our('k',k);
      end
    until J1='0'
   end;
 begin
   Xn:=0; Xk:=5;
   repeat
     PutA;
     Ou('Esc-exit,1-Nvr,2-Coefs,3-L,4-H,5-Xn,6-Xk,7-C,8-Sc,9-G,A-Groms');
     Str(Nvr,T10);    Ts:='Nvr='+T10;
     Str(L,T10);      Ts:=Ts+', L='+T10;
     Str(H,T10);      Ts:=Ts+', H='+T10;
     Str(Xn:1:3,T10); Ts:=Ts+', Xn='+T10;
     Str(Xk:1:3,T10); Ts:=Ts+', Xk='+T10;
     Str(C,T10);      Ts:=Ts+', C='+T10;
     Info; J:=ReadKey;
```

```
      case J of
       '1': Oui('Nvr(in[1..4]',Nvr);
       '2': Coefs;
       '3': Oui('L', L);
       '4': Oui('H', H);
       '5': Our('Xn', Xn);
       '6': Our('Xk', Xk);
       '7': Oui('C', C);
       '8': begin
             Xmin:=Xn; Xmax:=Xk;
             Ymin:=0; Ymax:=0;
             Dx:=(Xk-Xn)/L; Mo[-1]:=L;
             for s:=0 to L do
              begin
                y:=F(Xn+s*Dx); Mo[s]:=y;
                if y<Ymin then Ymin:=y;
                if y>Ymax then Ymax:=y
              end;
             X0Y0(false); Cleardevice; SystCoor;
             Graphic(Mo,C)
            end;
       '9': begin
             Dx:=(Xk-Xn)/L; Mo[-1]:=L;
             for s:=0 to L do
               Mo[s]:=F(Xn+S*Dx);
             Graphic (Mo,C)
            end;
       'a','A': Service
      end
    until J=#27;
    CloseGraph
  end.
```

Підключаємо до програми модулі Crt, Graph, Serv та Groms. З модулями Graph та Serv – зрозуміло: адже збираємось графіки малювати. А от відносно Crt – навіщо? А от на що. В Crt є такі корисні функції:

function KeyPressed: Boolean;

– повертає true, якщо на клавіатурі була натиснута клавіша (будь-яка), та false – в протилежному випадку; функція KeyPressed не затримує виконання програми.

function ReadKey: char;

– зчитує символ з клавіатури (код натисненої клавіші) без ехоповтору на екрані.

Призупиняє виконання програми до натиснення на будь-яку клавішу крім Shift, Ctrl, Alt, CapsLock, NumLock, ScrollLock.

Використання першої з двох щойно згаданих функції ми прибережемо на потім, а от функцією ReadKey збираємось активно користуватись в даній програмі для вибору опцій меню.
Далі в описовій частині програми Graphics іде ряд типізованих констант. Це, власне, змінні з початковими значеннями.
Nvr – номер варіанта; ми збираємось задіяти в програмі чотири варіанти функції (див. нижче).
A, B, k – коефіцієнти, що беруть участь у формулах для функції.
Початкові значення констант задані "на всяк випадок" – в процесі роботи з програмою користувач може задавати їх на свій розсуд.
Змінна Mo: Coefl – призначається для розміщення ординат функції (Mo – масив ординат). Ми її умисно назвали так же, як в п.1.1 для кращого усвідомлення її суті.
Підпрограма-функція F обчислює значення функції для поточного значення X в залежності від поточних значень глобальних параметрів Nvr, A, B, k.
Процедура Coefs організує інтерактивне введення значень параметрів A, B, k.
Процедура Coefs має змінну T1:String. В ній ми збираємось формувати текст формули, що відповідає поточному значенню Nvr. Змінна J1:char призначається для вибору елементів меню даної підпрограми. В принципі, можна було б скористатись для цього однойменною глобальною змінною того ж типу (модуль Serv) але, як кажуть, краще коли "усе своє ношу з собою".
Операторна частина підпрограми Coefs організована у вигляді циклу repeat, вихід з якого відбувається при J1='0'. Домовимось на майбутнє, що вихід із основної програми будемо здійснювати натисканням клавіші Esc (код 27), а виходити із всіляких підменю – натисканням клавіші '0'.
Тіло циклу repeat починається викликом процедури PutA (див. модуль Serv) – це очистка першого рядка екрану. Потім процедура Ou (модуль Serv) виводить текст підменю
'0 – exit, 1 – A, 2 – B,3 – k',
яке треба сприймати так: якщо треба вийти з підменю – натискайте клавішу '0', натискаючи якусь із клавіш '1', '2', '3' ви організуєте введення нових значень A, B чи k.
Але щоб мати уявлення про поточні значення цих параметрів (коефіцієнтів), їх значенння повинні відображуватись в інформаційному рядку в нижній частині екрана. Отже, перед викликом процедури Info (модуль Serv) треба сформувати текст інформаційного рядка в змінній Ts:string[80] (модуль Serv).
У графічному режимі процедури
procedure OutText (Text:string) та
procedure OutTextXY (X, Y:integer; Text:string)
можуть виводити на екран лише текст. Числову ж інформацію перж ніж включити її в текстову змінну Ts треба перетворити у текстову форму. Реалізується це процедурою Str (модуль System)
procedure Str (X[: width [: Decimals]]; var st: string);
Тут X – число, що перетворюється (дійсного або цілочислового типу).

Необов'язкові параметри Width та Decimals (обидва типу byte) – це ширина (точніше – довжина) тобто загальне число символів у тексті, в який має перетворитись число X, та кількість цифр після десяткової крапки у тексті, числа, що формується.
Зазвичай текстова форма складається із символу знака числа X (якщо X>0 – знак відсутній), окремої цифри (не нуль), крапки та Decimals цифр. При необхідності далі іде латинська буква E (еквівалент 10), після неї знак (якщо порядок, тобто степінь 10 додатній, то його знак може бути відсутнім), і нарешті, що представляє порядок.
Визначення параметра Width, як видно зі сказаного, може створювати проблеми – піди знай яку величину може, прийняти параметр, що перетворюється. Та й потім –"раз на раз не приходиться". На практиці, якщо компілятор виявить, що Width більше, ніж йому потрібно для відображення числа, то "зайве" позиції він перетворить на пробіли перед текстом, в протилежному випадку параметр Width ігнорується.
При перетворенні цілочислових X можна не задавати Width (і, зрозуміло, Decimals – на те вони й необовязкові, на що вказуєть квадратні дужки у визначенні процедури Str). Тоді компілятор формує текст мінімальної довжини, що якраз дуже до речі, коли формується текст інформаційного рядка, в який треба втиснути інформацію про багато параметрів.
Отже, формування Ts починається з перетворення в текст T10 значення A: real. Сам же текст Ts починається з тексту 'A=', до якого приєднується (+ символ конкатенації) T10. Далі до цього тексту приєднується тексти виду: кома, пробіл, ідентифікатор параметра, знак =, текст (значення параметра).
Має сенс інформацію про поточні значення, що входять в меню, розташовувати у тій же послідовності, в якій вони фігурують в меню, яке обслуговується інформаційним рядком – так зручніше розшукувати поточне значення потрібного параметра.
На завершення інформаційного рядка для меню процедури Coefs в нього добавляється текст формули функції,що відповідає поточному значенню Nvr.
Після того як Ts сформовано, воно виводиться на екран процедурою Info (модуль Serv). Отже, користувач має перед очима меню ('0 – exit, 1 – A, 2 – B,3 – k') та значення A, B, k в інформаційному рядку. Якщо він вважає за потрібне змінити (перезадати) значення будь-якого із представлених параметрів, він має натиснути відповідну клавішу (1 – для A, 2 – для B,3 – для k).
Елементи меню нумеруються щоб для вибору елемента (опції меню) можна було користуватись цифровою клавіатурою. Це зручно, оскільки цифрові клавіші не залежать від установленого драйвера шрифту (латинський чи кирилиця) від регістра (верхній чи нижній). Цифри легше відшукувати на клавіатурі, в них важче помилитись, а при користуванні цифровою клавіатурою – вони усі під рукою. Але інколи цифр не вистачає, тоді окремі елементи меню можна "нумерувати" буквами (як правило – латинськими оскільки драйвер латинського шрифту підключається за умовчанням).

Отже, свій вибір користувач реалізує натискаючи відповідну клавішу. Оператор J1:=ReadKey запам'ятовує в J1 код натиснутої клавіші, причому "вказівка" виконується "в одне натискання" (без додаткового натискання клавіші 'Enter'). Оператор case в залежності від натиснутої клавіші організує введення значення замовленого параметра.

Операторна частина основної програми починається із задання початкових значень (на всякий випадок) глобальних змінних (модуль Serv) Xn та Xk. Це робиться для того, щоб при першому формуванні інформаційного рядка для основного меню в якості значень цих параметрів не фігурували випадкові (можливо – абсурдні) числа. Далі в операторній частині програми фігурує конструкція repeat такої ж структури, яку ми щойно розглядали в підпрограмі Coefs. Взагалі структура циклу repeat з меню та відповідним йому інформаційним рядком, а також оператором-диспетчером case буде надалі використовуватись як типова для програм та підпрограм з меню.

Так от, згідно з основним меню програми, натискання на клавішу '1' ініціює введення нового значення Nvr – номера варіанту функції. Клавіша '2' – це виклик процедури Coefs. Опції '3 – L, 4 – H, 5 – Xn, 6 - Xk' – це відповідно введення L – ширини графіка в пікселях, H – його висоти (в пікселях), Xn – початкового та Xk – кінцевого значення X.

'7 – C' – це колір графіка (його код).

'8 – Sc' – формування системи координат.

Тут відбувається наступне. Мінімальне значення X тобто Xmin приймається рівним Xn, відповідно Xmax:=Xk. Таким чином, діапазон зміни X (від Xn до Xk) розтягується на всю ширину робочого поля (прямокутника L*H – див X0Y0 в модулі Serv). Мінімальне Ymin та максимальне Ymax для початку приймаються рівними нулю. Будемо вважати, що ми зацікавлені в тому, щоб вісь X (що відповідає нульовому значенню Y) була присутня на екрані.

Dx:=(Xk-Xn)/L;

– це крок зміни X, тобто його зміна на 1 піксель. Ми домовились (див. процедуру Graphic в модулі Serv) малювати графік у вигляді ламаної з кроком в 1 піксель по горизонталі (максимально детально). В комірку Mo[-1] заноситься L (процедура Graphic буде цікавитись вмістом цієї комірки). В циклі по S заповнюється масив Mo:CoefL і заодно визначається Ymin та Ymax в межах графіка, що формується.

Процедура X0Y0 (модуль Serv) розраховує параметри системи координат під графік. ClearDevice очищує екран - оскільки оновлюється система координат, то зображення, сформовані в старій системі координат, втрачають сенс – необхідно звільнити робоче поле під новий графік (графіки). Процедура SystCoor (модуль Serv) малює осі системи координат, а процедура Graphic (з того ж таки модуля Serv) – формує сам графік.

Можна було б, звичайно, саме малювання гшрафіка притримати для опції '9 – Gr', але якщо ми уже сформували систему координат під даний графік, то навіщо відкладати?

Взагалі-то винесення операції формування в окрему опцію '9 – Gr' пояснюється таким міркуванням. Можливо, користувач захоче на одному

рисунку (в одній системі координат) сформувати не один, а кілька графіків, наприклад, для вивчення впливу якогось із параметрів (скажімо, Nvr, A, B, чи k). Тоді задаючи для кожного значення параметра відповідне значення C (кольору) можна кожному значенню параметра поставити у відповідність свій графік (своїм кольором). Але тут треба бути уважним: в одній системі координат не можна варіювати Xn, Xk, L, H, оскільки вони впливають на параметри системи координат.

Якщо треба стерти сформоване зображення – клавіша '8' виконує ваш задум. Те, що при цьому ви заставите комп'ютер заново розраховувати параметри системи координат – вас це може не турбувати – для сучасного комп'ютера це така дрібниця, ви й не помітите, нова система координат з останнім варіантом графіка з'явиться перед вашими очима практично миттєво.

Приклади графіків, сформованих програмою Graphics, показані на рис.1.

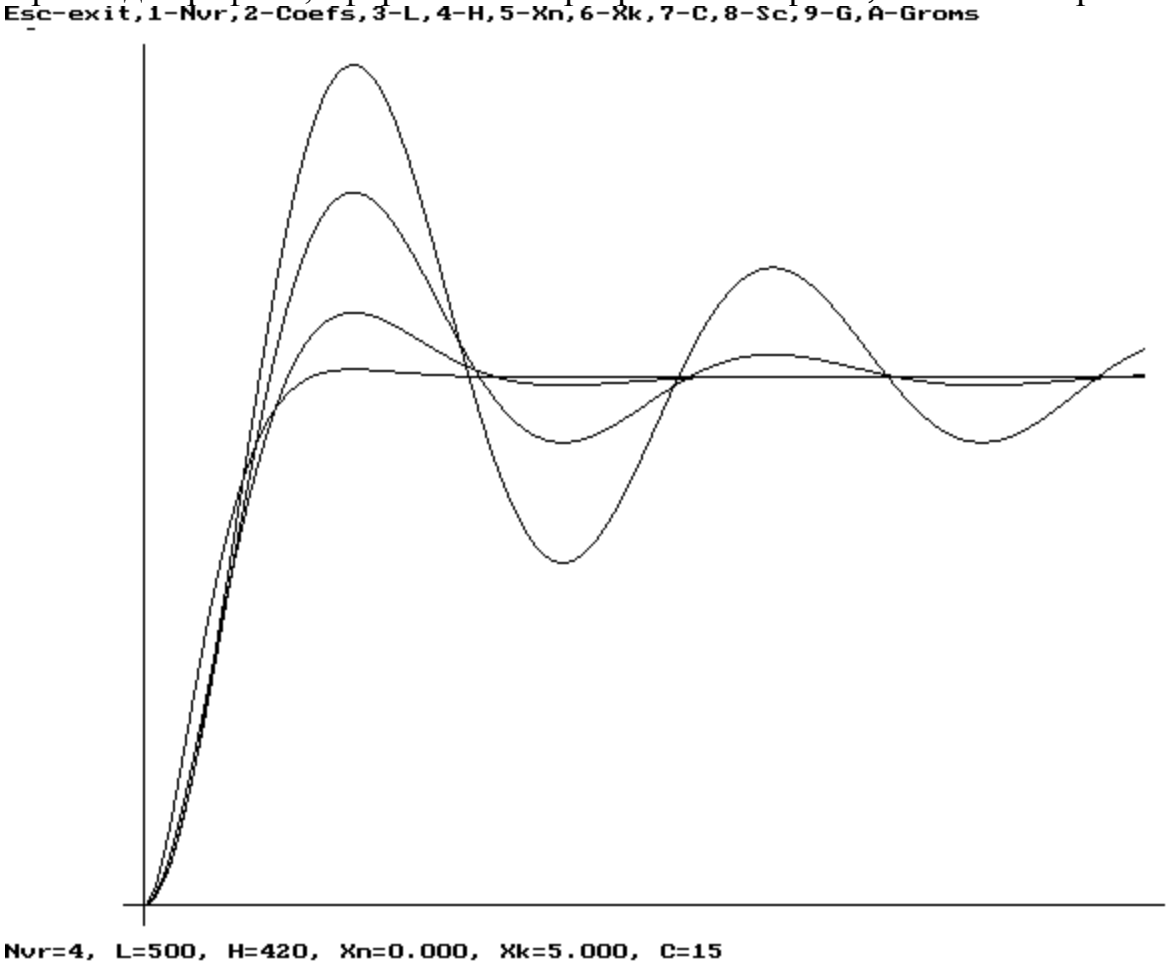

Рис.2. Приклад побудови графіка

Годографом називається графічне відображення залежності між двома змінними, скажімо, X та Y, що задані параметрично

$$\begin{cases} X = f_1(\omega), \\ Y = f_2(\omega), \end{cases}$$

де $f_1(\omega)$ та $f_2(\omega)$ – деякі функції параметра ω.

У загальному випадку ці функції можуть задаватись і неявно, наприклад шляхом реалізації якого завгодно алгоритму. В подальшому будемо орієнтуватися на те, що глобальні змінні X, Y: real (визначені в Serv) отримують свої значення в функції від поточного значення параметра w : real (програмний еквівалент параметра ω) за допомогою процедури UrGod. Ну наприклад:

```
procedure UrGod;
  var Xe, Ye, Wr: real;
   begin
       case Nvr of
           1: begin
                   X:= R0*cos(w);
                   Y:= R0*sin(w);
              end;
           2: begin
                    X:= R1*cos(w);
                    Y:= R0*sin(w);
              end;
           3: begin
                    Xe:= R0-R1; Ye:= Xe*w/R1;
                    X:= Xe*cos(w)-U*cos(Ye);
                    Y:= Xe*sin(w)-U*sin(Ye);
              end;
           4: begin
                    Xe:= R0+R1; Ye:= Xe*w/R1;
                    X:= Xe*cos(w)-U*cos(Ye);
                    Y:= Xe*sin(w)-U*sin(Ye);
              end;
           5: begin
                    Ur:= U*Pi/180;
                    Xe:= R1*cos(w); Ye:= R0*sin(w)
                    X:= Xe*cos(Ur)-Ye*sin(Ur);
                    Y:= Ye*cos(Ur)+Xe*sin(Ur);
              end;
       end
  end;
```

Тут параметри W:real; R0,R1,U:integer також глобальні, але визначаються (описуються) вони в основній програмі. Глобальним (з основної програми) є також параметр Nvr:byte (або integer) – номер варіанта. У нас тут в процедурі UrGod передбачено 5 можливих варіантів, але користувач може цей список змінити (замінити,розширити) на свій розсуд.

Процедура UrGod формується під розв'язувану задачу і природньо, що набір її параметрів (як локальних так і глобальних) має відповідати цій задачі. У наведеному варіанті процедури UrGod маємо:

Nvr=1. Це коло радіуса R0 (для зручності вважаємо, що R0 задається в пікселях), звідси і R0:integer.
Nvr=2. Це прямий еліпс (з осями паралельними осям системи координат хоу). Півосі еліпса – R1 (по горизонталі) та R0 (по вертикалі).
Nvr=3. Це гіпоциклоїда, у якої R0- радіус нерухомого кола, R1 – радіус колеса, що котиться всередині нерухомого без ковзання, U – довжина штанги, закріпленої на рухомому колесі (точніше – це відстань від центра рухомого колеса до точки фіксації пера на штанзі, саме це перо і малює гіпоциклоїду).
Nvr=4. Це епіциклоїда. Тут рухоме колесо котиться зовні по нерухомому.
Nvr=5. Нахилений еліпс, де U – кут повороту еліпса навколо його осі. Кути в Турбо Паскалі прийнято задавати в градусах, саме тому U: integer.
Параметр w це кут (в радіанах) повороту радіуса-вектора для Nvr=1,2,5
та кут нахилу лінії, що з'єднує осі нерухомого та рухомого коліс для Nvr=3,4.
За центр усіх перерахованих фігур приймається початок координат (X0,Y0).

Формули для нахиленого еліпса будуть виведені в п. 1.7.

Перше, що треба підготувати для малювання годографа, – це сформувати систему координат. Приймемо, що годограф, що формується, повинен по можливості повно використовувати відведений під нього прямокутник L x H. Це треба розуміти так, що він має одночасно дотикатись або до верхньої і нижньої, або до правої і лівої сторін вибраного прямокутника.
Діапазон зміни параметра W може або "набиватись" (скажімо 0..2π для повного кола в Nvr =1 або повного еліпса в Nvr = 2) або ж користувач може замовити не всю фігуру, а лише її дугу . Для епі- та гіпоциклоїд в залежності від співвідношення R0 та R1 можуть представляти інтерес самі різні діапазони. Отже, будемо задавати діапазон w двома параметрами: Wn та Wk – початковим та кінцевим значеннями w (в радіанах). Поточне значення W та крок Dw (обидва типу real) – також глобальні.
Для визначення Xmin, Xmax, Ymin, Ymax будемо сканувати діапазон Wn..Wk з кроком Dw := (Wk-Wn)/Nsc, де Nsc (глобальний параметр типу integer або word) – кількість кроків у межах діапазону. Тут варто зробити таке зауваження. Існують годографи, на яких мітки значень параметра розподіляються надзвичайно нерівномірно (ну, наприклад, грубо кажучи, перша половина годографу відповідає діапазону 0..0,1, а друга – 0,1..∞).
Тому для "надійного" обстеження заданного діапазону треба досить акуратно підбирати Nsc, а отже і крок Dw. Але це приходить з досвідом, зокрема, з досвідом роботи з конкретним типом годографа.
Оформимо процес сканування (розвідки) у вигляді підпрограми Scan.

```
procedure Scan;
  var S : integer;
      J : char;
  begin
     Dw:= (Wk-Wn)/Nsc;
     Xu:= (GetMaxX-Nsc) div 2;
     PutA; Rectangle (Xu,0,Xu+Nsc,5);
```

```
  Ou( '0-exit '); S:= 0; J:= '1';
  Xmin:= 0; Xmax:= 0; Ymin:= 0; Ymax:= 0;
repeat
    w:= Wn+S*Dw; UrGod;
    Line(Xu+s,1,Xu+s,4);
    if  X<Xmin then Xmin:=X;
        if  X>Xmax then Xmax:=X;
    if  Y<Ymin then Ymin:=Y;
        if  Y>Ymax then Ymax:=Y;
        inc(S);
        if KeyPressed then J:= ReadKey;
    until  (S>Nsc) or (J= '0' );
    X0Y0(true)
 end;
```

Механізм функціонування підпрограми Scan досить очевидний. Відзначимо дві обставини.

Перша. У загальному випадку процедура UrGod може бути самої різної трудомісткості в залежності від розв'язуваної задачі. І значення Nsc, про яке ми щойно говорили, також може бути задане користувачем досить значним (піди знай що йому прийде в голову ?). Таким чином, скажімо, при несприятливих умовах процес сканування може затягтись. Аж до того, що у користувача може скластись враження про те, що комп'ютер "завис". Щоб прояснити ситуацію у Scan передбачено такий механізм.

Обчислюється значення глобальної змінної

```
Xu:= (GetMaxX-Nsc) div 2;
```

Тут Xu - це горизонтальна екранна координата лівого верхнього кута прямокутника шириною Nsc пікселів, який (після очистки для нього місця процедурою PutA) тут же малюється у верхній частині екрану. Xu центрує прямокутник по горизонталі. Щоб прямокутник не вийшов за межі екрану Nsc має не перевищувати 600 (вважаємо, що GetMaxX = 639). Висота прямокутника 5 пікселів. Отже, ширина прямокутника в пікселях дорівнює Nsc. В циклі repeat при виконанні кожного кроку сканування прцедура Line зафарбовує чергову вертикальну полоску шириною в 1 піксель в межах намальованого прямокутника, поступово зафарбовуючи його.

Тим самим користувач отримує можливість простежити темп просування процесу сканування, оцінювати скільки ще залишилось часу до його завершення. Якщо, на думку користувача процес надто затягується (наприклад, внаслідок невдалого задання діапазону та (чи) Nsc), то натисканням на клавішу '0' він може достроково завершити процес (цей вихід йому люб'язно підказує виведений у лівому верхньому куті екрана (перед прямокутником) текст '0-exit ' (треба вийти – натисни '0')).

До речі, може виявитись, що інформація, отримана в результаті хай і не завершеної розвідки, достатня для формування адекватної системи

координат. Ну, а коли ні, то після коригування діапазону та Nsc процес можна і повторити.

За натисканням (ненатисканням) клавіш слідкує оператор

if KeyPressed then J:= ReadKey;

Якщо ніхто нічого не натискає, то функція KeyPressed повертає значення false – і нічого не відбувається (виконання підпрограми Scan не зупиняється – процес сканування продовжується). Але тільки-но якась клавіша виявиться натиснутою функція KeyPressed стає ріною true і оператор

J:= ReadKey

запам'ятовує в змінну J:char код натиснутої клавіші (з подачі функції ReadKey). І якщо цією клавішею виявиться саме клавіша '0' (як домовились '0-exit '), то логічний вираз, що стоїть після until, виявиться рівним true – відбудеться вихід (достроковий) з циклу repeat. Щоб цикл не перервався уже після першого циклу сканування (це в тому випадку коли б J випадково виявилось би рівним '0'), перед входом в цикл змінній J присвоюється значення '1' (аби не '0').

Природним виходом з repeat буде завершення процесу сканування (S>Nsc).

І друге. Початкові значення Xmin, Xmax, Ymin, Ymax перед початком сканування приймають рівними нулю. Це тому, що ми хотіли щоб початок системи кординат, що формується, був присутнім на екрані (в межах заявленого прямокутника L x H). Якщо у користувача (програміста) є на цей рахунок інші міркування, то треба поміняти відповідний рядок підпрограми.

Завершує підпрограму процедура

X0Y0(true),

яка визначає параметри системи координат XOY, заради чого, власне, і "город городили".

Після того як система координат сформована (байдуже, намальовані її осі координат на екрані чи ні – комп'ютер параметри системи пам'ятає) – можна приступити до малювання годографа.

Можна запропонувати ряд алгоритмів формування кривої.

Першим, певне, приходить у голову такий алгоритм – це ламана з фіксованим кроком Dw по параметру w. Це щось на зразок щойно розглянутого сканування, треба тільки доповнити його малюванням на кожному кроці відповідного відрізка ламаної. Але, як уже раніше згадувалось, імовірна велика нерівномірність міток параметра вздовж годографа. І тоді може виявитись, наприклад, що на початковій ділянці кривої відрізки ламаної будуть надто довгими (при чому деталі годографа будуть втрачатись), а в кінці відрізки зіллються в один піксель. Можна пробувати збільшити Nsc (чи його аналог – кількість відрізків ламаної), покращуючи тим самим якість початкової ділянки годографа. Можна, звичайно, варіювати діапазон w і навіть, при потребі, розбивати на ряд піддіапазонів з різними кроками на кожному з них.

Але давайте приймемо до уваги ще одне міркування. Відомо, що дія стандартної процедури SetWriteMode(1) не розповсюджується на криволінійні фігури, отже, стандартними засобами Турбо Паскалю малювати

годограф не виходить. А чому б нам по ходу не заповнити цей пробіл? Так що, давайте поставимо перед собою задачу розробити такий алгоритм, в якому кожен піксель годографа зафарбовується один (і тільки один!) раз. Тоді при необхідності його (піксель, а в результаті і весь годограф) можна було б малювати інверсним кольором. І це при тому (природне бажання), що годограф має бути суцільним.

Отже, припустимо, що ми маємо якийсь піксель, що належить годографу (перший або, в принципі, будь-який поточний). І до цього пікселя треба домальовати черговий (наступний). Розглянемо схему на рис.3.

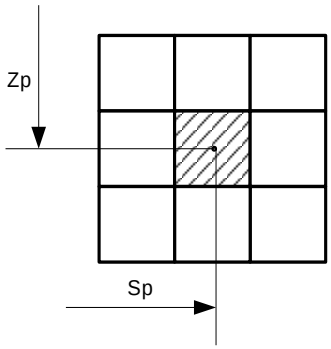

Рис.3 Схема поточного пікселя годографа та його найближчого оточення.

Заштрихований прямокутник зображує наявний піксель годографа з екранними координатами (Sp, Zp), де Sp – горизонтальна, Zp – вертикальна координата. Решта прямокутників – найближче оточення заштрихованого.

Щоб виконати умову неперервності годографа – черговим його пікселем має бути один з незаштрихованих його претендентів.

Перш ніж продовжувати розгляд алгоритму оформимо у вигляді процедури SZ перерахунок значень X,Y чергової точки годографа (чи претендента на цю роль) в її екранні координати S,Z.

```
procedure  SZ;
  begin
      UrGod;
       if Dx <>0  then  S:= X0+round(X/DX)
                   else   S:= X0;
          if  Dy <>0 then  Z:= Y0-round(Y/DY)
                       else   Z:= Y0;
   end;
```

Тут процедура UrGod поставляє значення X,Y (глобальні) відповідні поточному значенню параметр W для годографа.

А тепер процедура PointGod, що домальовує черговий піксель годографа.

```
procedure  PointGod;
    const Kw=1.2;
    var Mdx, Mdy,Md,Sp,Zp : integer;
        Wp: real;
```

```
begin
    Wp:=W; Sp:=S; Zp:=Z;
    repeat
        W:=Wp+Dw; SZ;
        Mdx:= abs(S-Sp); Mdy:= abs(Z-Zp);
        if  Mdx>Mdy then Md:= Mdx
                         else  Md:=Mdy;
        if  Md<>1 then
            if Md>1 then Dw:=Dw/Md
                    else  Dw:= Dw*Kw
    until (Md = 1) or (W>Wk);
  if Md = 1 then
    if C >= 0 then PutPixel(S,Z,C)
              else  PutPixel(S,Z,GetMaxColor-
                                 GetPixel(S,Z) )
end;
```

А тепер детальніше. Отже, запам'ятовуємо параметри відправної точки (Wp:=W; Sp:=S; Zp:=Z). Входимо в цикл repeat.

До значення Wp добавляємо Dw - отримуємо нове значення W та викликаємо процедуру SZ. Маємо нові S та Z. Визначаємо Mdx – відстань по горизонталі та Mdy – по вертикалі (в пікселях) між відправним пікселем (Sp, Zp) та отриманим (S, Z). Знаходимо більшу з цих відстаней Md.

Якщо Md не дорівнює 1 (умова дотикання пікселів (Sp, Zp) та (S, Z)), то одне з двох: або Md>1, або Md =0. При Md>1 відправний (Sp, Zp) та поточний (S, Z) пікселі знаходяться надто далеко один від одного (значить, ми перебрали з Dw, і його треба зменшити – пропонується поділити його на Md) інакше (Md = 0) Dw треба збільшити (ми його множимо на Kw = 1.2).

Але чому саме на 1.2? Справа от в чому. Якщо коефіцієнт Kw (константа в PointGod) взяти занадто великим, то може статись, що замість ситуації Md = 0, множачи Dw на Kw можемо отримати ситуацію Md>1, з котрої можемо повернутись у ситуацію Md = 0 і т.д. – проглядається перспектива зациклювання. Чим менше Kw, тим зациклювання є менш імовірним, зате, щоб "вибратись" з Md = 0, можливо, знадобиться кілька спроб – затягування процесу. Не варто забувати також про особливості годографа, що будується (вони бувають такі різні!). Отже, коефіцієнт Kw варто розглядати як настроювальний параметр – користувач може (і повинен) його встановити так, щоб він найкращим чином відповідав особливостям його "улюблених" годографів.

Як щойно описано, у циклі repeat підбирається таке значення Dw (а, отже, і w), при якому Md = 1. Коли це так – вихід із циклу і малювання знайденого пікселя (при C < 0 інверсним кольором). На кожній ітерації оновлюється і w. Якщо w > wk (кінцевого у заданому діапазоні) – робота по побудові годографа завершується. А коли так – вихід з циклу.

Якщо описану процедуру PointGod помістити в цикл, який заставить її послідовно малювати піксель за пікселем, то в кінцевому результаті можна

сформувати і весь годограф цілком. Швидкодіючий комп'ютер (при відносно мало трудомісткій процедурі UrGod, при незначній ширині діапазону w і т.д.) сформує замовлений годограф за частки секунди або, скажімо, за одиниці секунд. Ну, а якщо пікселів у годографа багато, а на пошук кожного з них витрачається багато часу – процес може набагато (або навіть недопустимо) затягнутись.

Справа посувалась би швидше аби ми не наполягали на максимальній акуратності (детальності) годографа (піксель за пікселем) як це робиться в PointGod, а згодились би на представлення годографа ламаною але обмежили б максимальну (можливо заодно – і мінімальну) довжину кожного відрізка ламаної. Ну, скажімо, поставили б умову щоб довжина кожного відрізка не виходила б з діапазону Min..Max, де, наприклад, Min = 5, Max = 15 пікселів.

В загальному випадку Min,Max – настроювальні параметри. Ця ідея реалізується процедурою LineGod.

```
procedure  LineGod;
    var Mdx, Mdy, Md, Sp, Zp : integer;
        Kw, Sr, Wp : real;
    begin
          Wp:= W; Sp:= S; Zp:= Z;
          Sr:= (Min+Max)/2; Kw:= Max/Min/2;
          repeat
               W:= Wp+Dw; SZ;
               Mdx:= abs(S-Sp); Mdy:= abs(Z-Zp);
               if Mdx > Mdy then Md:= Mdx
                                else Md:= Mdy;
               if (Md > Max) or (Md < Min) then
                  if Md > Max then Dw:= Dw*Sr/Md;
                                else  Dw:= Dw*Kw;
           until ((Md <= Max) and (Md >= Min)) or (w>wk);
           if (Md <= Max)   and (Md >= Min) then
              begin
                  Line(Sp,Zp,s,z);
                  if C < 0 then PutPixel (s,z,GetMaxColor-
                                          GetPixel(s,z))
          end;
    end;
```

Легко бачити, що алгоритм в LineGod майже не відрізняється від такого в PointGod. Відмінність в коефіцієнтах Sr та Kw, причому останній уже не константа, а визначаться в функції від Min та Max. Але це – деталі. При C < 0 (інверсний колір) стики ламаної зафарбовуються додатково.

Процедура Godo, лістинг якої наводиться нижче, "керує" формуванням годографа в цілому.

```
procedure  Godo;
    var J : char;
    begin
```

```
                Dw:= (Wk-Wn)/Nsc; W:= Wn; SZ;
                if C >= 0
                   then
                    begin
                        SetColor(C); PutPixel(s,z,C)
                    end
                    else
                      case Tg of
                          'P': PutPixel (s,z,GetMaxColor-GetPixel(s,z));
                          'L': SetWriteMode(1);
                      end;
              J:= '1'; PutA; Ou('0-exit'); OutTextXY(434,0, 'w=');
              repeat
                   case Tg of
                     'P' : PointGod;
                     'L' : LineGod;
                   end;
                    Str(W:1:3,T10); Clear(450,0,GetMaxX-1,12);
                    OutTextXY(450,0,T10);
                    if KeyPressed then J:=ReadKey
              until (w>wk) or (J= '0');
              if C >= 0 then SetColor(15)
                          else SetWriteMode(0)

          end;
```

Тут глобальна змінна Tg : char (описується в основній програмі) задає тип годографа (Tg = 'P' – з використанням процедури PointGod, та Tg = 'L' – процедури LineGod). Решта очевидне.

Процедури Scan, SZ, PointGod, LineGod та Godo доцільно об’єднати в один файл (назвати його, скажімо, Godogr) з тим щоб при необхідності через команду компілятору {$i Godogr.pas} підключати його до описової частини програми (після процедури UrGod, послугами якої користується SZ).

Текст файла Godogr наводиться у додатку 3.

А тепер наведемо текст демонстраційно-відлагоджувальної програми Godograf

```
Program Godograf;
    uses Crt,Graph,Serv;
    const Nvr:integer = 1;
          R0: integer = 50;
          R1: :integer = 20;
          U: :integer = 10;
          Nsc:integer = 100;
          Wn: real = 0;
          Wk: real = 6.28;
    var w, Dw: real;
```

```
        procedure UrGod;
          var Xe,Ye,Ur: real;
            begin
              case Nvr of
                1: begin
                     X:= R0*cos(w);
                     Y:= R0*sin(w);
                   end;
                2: begin
                     X:= R1*cos(w);
                     Y:= R1*sin(w);
                   end;
                3: begin
                     Xe:= R0-R1; Ye:= Xe*w/R1;
                     X:= Xe*cos(w)-U*cos(Ye);
                     Y:= Xe*sin(w)-U*sin(Ye);
                   end;
                4: begin
                     Xe:= R0+R1; Ye:= Xe*w/R1;
                       X:= Xe*cos(w)-U*cos(Ye);
                       Y:= Xe*sin(w)-U*sin(Ye);
                     end;
                5: begin
                     Ur:= U*Pi/180;
                     Xe:= R1*cos(w); Ye:= R0*sin(w)
                     X:= Xe*cos(Ur)-Ye*sin(Ur);
                     Y:= Ye*cos(Ur)+Xe*sin(Ur);
                   end
              end
          end;
 {$i Godogr.pas}
begin
  repeat
    PutA; Ou('Esc-exit, 0-Nvr, 1-R0, 2-R1, 3-U, 4-Wn, 5-wk, 6-C, 7-Scan,
              8-Godo, 9-Sc');
    Str(Nvr,T10);                    Ts:= 'Nvr= ' +T10;
    Str(R0,T10);                     Ts:= Ts+ ',  R0=' +T10;
    Str(R1,T10);                     Ts:= Ts+ ',  R1=' +T10;
    Str(U,T10);                      Ts:= Ts+ ',  U=' +T10;
    Str(Wn:1:3,T10);                 Ts:= Ts+ ',  Wn=' +T10;
    Str(Wk:1:3,T10);                 Ts:= Ts+ ',  Wk=' +T10;
    Str(C,T10);                      Ts:= Ts+ ',  C=' +T10;
    Info; J:= ReadKey;
    case J of
      '0' : Oui('Nvr', Nvr);
```

```
        '1' : Oui('R0', R0);
        '2' : Oui('R1', R1);
        '3' : Oui('U', U);
        '4' : Our('Wn', Wn);
        '5' : Our('Wk', Wk);
        '6' : Oui('C', C);
        '7' : Scan;
        '8' : Godo;
        '9' : SystCoor
      end
  until J = #27;
  CloseGraph
end.
```

На рис.4 показано приклади годографів, сформованих даною програмою.

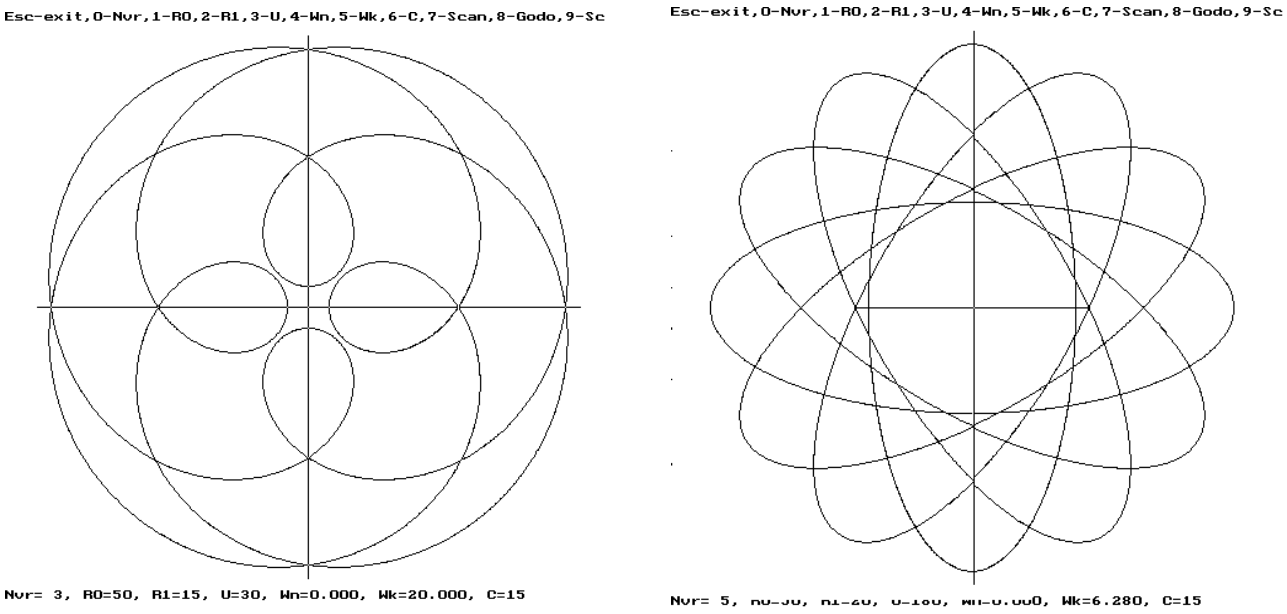

Рис.4. Приклади формування годографів програмою Godograf.

Лекція 2. Додавання, множення та ділення поліномів.

Поліноми – дуже зручний інструмент в математиці. З одного боку – що може бути простіше від полінома? З іншого – це досить гнучкий інструмент. Нарощуючи степінь полінома можна розширювати його можливості чи не до нескінченності. Поліноми можна використовувати для цілей інтерполяції, екстраполяції, згладжування. Кусочно-поліноміальні конструкції – сплайни та B-сплайни можна розглядати як високоякісний з широкими можливостями апарат для формування графічних зображень, для керування станками з цифровим програмним керуванням. Характеристичні поліноми, що відповідають диференціальним рівнянням та їх системам, несуть масу цінної інформації про досліджувані системи. Список “заслуг” поліномів можна

продовжувати. Тому бажано розглянути принаймні найпростіші операції над поліномами, до чого ми зараз і переходимо.
$A(x)=a_0+a_1x+a_2x^2+\ldots+a_nx^n$,

Коефіцієнти яких зберігаються в масиві типу Coef за такою схемою

-1	0	1	2	3	…	n	…	31
n	a_0	a_1	a_2	a_3	…	a_n		

A:Coef

Алгебраїчне додавання поліномів

$C(x)=A(x)\pm B(x)$,

де $A(x)=a_0+a_1x+a_2x^2+\ldots+a_{Na}x^{Na}$,
$B(x)=b_0+b_1x+b_2x^2+\ldots+b_{Nb}x^{Nb}$,

Додати два поліноми - це значить додати (з відповідними знаками) коефіцієнти при однакових степенях аргументу (у нашому випадку x). Це можна зробити так:

```
procedure AlSumPol(A,B:coef; OP:char; var C:coef);
var Na,Nb,Nc,S:integer;
begin
      Na:=round(A[-1]); Nb:=round(B[-1]);
      if Na>Nb     then Nc:=Na
                   else Nc:=Nb;
      C[-1]:=Nc;
      if Na<Nc then
            for S:=Na+1 to Nc do A[S]:=0;
      if Nb<Nc then
            for S:=Nb+1 to Nc do B[S]:=0;
      if OP='+'     then C[S]:=A[S]+B[S]
                   else C[S]:=A[S]-B[S]
end;
```

Якщо алгебраїчно додаються два поліноми різних степенів ,то , очевидно що поліном – сума буде мати більший із степенів поліномів – доданків. Саме таким чином і визначається Nc у підпрограмі AlSumPol. Далі поліном – доданок з меншим степенем формально доводиться до степеня Nc за рахунок старших доданків з нульовими значеннями коефіцієнтів. Параметр OP: char може приймати одне з двох можливих значень: OP='+' якщо поліноми додаються, та OP='-', якщо поліном $B(x)$ вирахувується з полінома $A(x)$. Саме ця операція алгебраїчного додавання реалізується в останньому циклі підпрограми.
В порядку розширення можливостей операції можна було б розглянути операцію

$C(x)=Ka\cdot A(x)+ Kb \cdot B(x)$

де Ka,Kb- множники типу real. Програмна реалізація операції – останній цикл в підпрограмі набув би такого вигляду

```
for s:=0 to Nc do
C[s]:=Ka*A[s]+Kb*B[s].
```

Множення поліномів

$$C(x)=A(x)*B(x)$$

При множенні поліномів їх степені додаються, а коефіцієнти полінома – добутку визначаються додаванням коефіцієнтів при однакових степенях добутків кожного члена полінома A(x) на кожен член полінома B(x). Отримуємо підпрограму UmnPol.

```
procedure UmnPol (A,B:Coef; var C: Coef );
          var  Na, Nb, Nc, s, z: integer;
           begin
                Na:=round(A[-1]); Nb:=round(B[-1]);
                Nc:=Na+Nb;   C[-1]:=Nc;
                for s:=0 to Na  do  C[s]:=0;
                for z:=0  to  Na  do
                for s:=0  to  Nb do
                    C[z+s]:=C[z+s]+A[z]*B[s]
           end;
```

Якщо треба помножити поліном A(x) на коефіцієнт k:real, то цей коефіцієнт формально можна розглядати як поліном B(x) нульового степеня, у якого B[0]=k, і тим самим ця задача може розв’язуватись процедурою UmnPol, так що необхідність в окремій процедурі, яка б множила поліном на коєфіцієнт - відпадає. З іншого боку, прооцедуру UmnPol можна використовувати для піднесення полінома до цілого додатнього степеня, наприклад, n:byte.

```
procedure PolStn(A:Coef; n:bute; var C:Coef);
     var  s:integer;
          B:Coef;
     begin
          C[-1]:=0;  C[0]:=1;
          for  s:=1  to n  do  UmnPol(C,A,C)
     end;
```

При n=0 поліном C(x)=1 - за визначенням.

Ділення поліномів

$$\frac{A(x)}{B(x)} = C(x) + \frac{D(x)}{B(x)}.$$

Тут

A(x)-поліном-ділене (степеня Na),

B(x)-поліном-ділитель (степеня Nb),

C(x)-поліном-частка(Nc),

D(x)-поліном-остача(Nd).

Якщо степінь Na полінома A(x) менше степеня Nb полінома B(x), то C(x)=0 і D(x)=A(x). Якщо Na>Nb, то Nc=Nb-Na, Na$\le$Nb-1. Щоб розробити алгоритми формування поліномів C(x) та D(x) при Na>Nb розглянемо приклад з метою виявлення закономірностей, які потім треба буде узагальнити на загальний випадок. Отже нехай Na=6, Nb=2, тоді формулу можна представити в розгорнутому вигляді

$$\frac{a_6x^6+a_5x^5+a_4x^4+a_3x^3+a_2x^2+a_1x+a_0}{b_2x^2+b_1x+b_0}=c_4x^4+c_3x^4+c_2x^2+c_1x+c_0+\frac{d_1x+d_0}{b_2x^2+b_1x+b_0}$$

Помножимо на $B(x)=b_2x^2+b_1x+b_0.$

Отримаємо

$$a_6x^6+a_5x^5+a_4x^4+a_3x^3+a_2x^2+a_1x+a_0=(c_4x^4+c_3x^3+c_2x^2+c_1x+c_0)(b_2x^2+b_1x+b_0)+d_1x+d_0.$$

А тепер розкриємо дужки та порівняємо коєфіцієнти при однакових степенях x в правій та лівій частинах останнього виразу.

x^6: $C_nb_2=a_6,$

x^5: $C_nb_1+C_3b_2+C_2b_2=a_4,$

x^4: $C_nb_0+C_3b_1+C_2b_2=a_3,$

x^3: $C_3b_0+C_2b_1+C_1b_2=a_3,$

x^2: $C_2b_0+C_1b_1+C_0b_2=a_2,$

x^1: $C_1b_0+C_0b_1+d_1=a_1,$

x^0: $C_0b_2+d_0=a_0.$

Виписану систему співвідношень можна розглядати як систему алгебраїчних рівнянь відносно шуканих коефіцієнтів поліномів C(x) та D(x). Причому, що суттєво, дана система елементарно розв'язується шляхом послідовного визначення невідомих: з першого рівня визначаємо C_1, з другого C_3, і так далі до C_0, з предостатнього - d_1 та з останнього -d_0.

Якщо прийняти C_5=0 та $C_6=0$ (що так і є насправді), тобто формально довести степінь C(x) до степеня полінома A(x), то алгоритм послідовного визначення коєфіцієнтів полінома C(x) можна узагальнити у вигляді очевидної формули

$$Cz=\frac{a_{Nb}-\sum_{s=0}^{nd} b_s c_{Nb+z-s}}{b_{Nb}}, \qquad Nc\ge z\ge 0$$

Тут z послідовно приймає значення Nc, Nc-1, Nc-2,…,0.

Для визначення коєфіцієнтів полінома D(x) за умови, що коєфіцієнти C(x) уже визначені, справедлива формула

$$d_z=a_z-\sum_{s=0}^{z} b_s c_{z-s}, \qquad 0\le z\le Nd.$$

При користуванні формулою процес перебору z може бути довільним.

З врахуванням сказаного операцію можна оформити у вигляді процедури DelPol.

```
procedure DelPol(A, B:Coef; var C, D:Coef);
     var z, s, Na, Nb, Nc, Nd: integer;
       Sum: real;
       begin
           Na:= round(A[-1]);   Nb:= round(B[-1]);
           if Na<Nb
             then
              begin
                    D:=A;  C[-1]:=0;   C[0]:=0
               end
              else
              begin
                    Nd:=Nb-1;    Nc:=Na-Nb;
                    for z:=Na+1  to  Na  do
                      C[z]:=0;
                    for z:=Nc  downto 0 do
                     begin
                          Sum:=0;
                          for s:=0 to Nd  do
                            Sum:=Sum+B[s]*C[Nb+z-s]
                     end;
                  for z:=0 to  Nd  do
                    begin
                         Sum:=0;
                         for s:=0 to z  do
                         Sum:=Sum+B[s]*C[z-s];
                         D[z]:=A[z]-Sum
                     end;
             C[-1]:=Nc;  D[-1]:=Nd
    end;
end;
```

Поліном A(x) може в принципі, ділитись на B(x) і без остачі. В загальному випадку поліном D(x) може мати степінь в діапазоні $0 \le Nd \le Nb-1$.

Втрата степеня може мати місця і при алгебраїчному додавані поліномів одинакових степенів, у яких старші коєфіцієнти можуть компенсувати один одного при їх алгебраїчному додаванні. При множенні поліномів коли один із множників-нуль, то і добуток буде нулем, хоча формально степінь полінома-добутку буде дорівнювати степеню ненульового співмножника. Необхідно також не забувати про те, що поліноми, якими ми користуємось в якості операндів, можуть бути результатом реалізації будь-яких алгоритмів, а тому можуть мати які завгодно коефіцієнти, в тому числі і нульові (і для старших членів також). Звідси випливає, що формальні степені полінома можуть не співпадати з їх фактичними степенями. Справа ускладнюється ще й тим, що

значення коефіцієнтів (справжніх, «живих», а не придуманих для прикладу) відомо лише наближено. І відрізнити справжній нуль від «віртуального» може виявитись не так просто (а то й неможливо). Домовимось вважати старший коефіцієнт полінома практично рівним нулю, якщо його модуль не перевищує деякої наперед заданої малої величини ε.

Розвинення відношення поліномів у степеневий ряд

$$\frac{A(x)}{B(x)} = e_0 + e_1 x + e_2 x^2 + ... + e_n x^n + ...$$

Будь-яких обмежень на степінь поліномів *A(x)* та *B(x)* накладати не будемо. Якщо поліном *A(x)* ділиться без остачі на *B(x),* то отримаємо поліном скінченного степеня. В загальному ж випадку ряд *E(x)* – нескінченний. Для практичних потреб ми будемо його обмежувати до полінома *n*-го степеня, де *n*- задається користувачем (це диктується умовами задачі, яка розв'язується). Якщо порівняти та , то ряд *E(x)* в можна розглядати як суму полінома *C(x)* та ряду, в який розкладається *D(x)/B(x).* Запишемо в розгорнутому вигляді

$$\frac{a_0 + a_1 x + a_2 x^2 + ... + a_{Na} x^{Na}}{b_0 + b_1 x + b_2 x^2 + ... + b_{Nb} x^{Nb}} = e_0 + e_1 x + e_2 x^2 + ... + e_n x^n + ...$$

та помножимо останнє співвідношення на знаменник лівої частини. Отримаємо

$$(e_0 + e_1 x + e_2 x^2 + ... + e_n x^n + ...)(a_0 + a_1 x + a_2 x^2 + ... + a_{Na} x^{Na}) = b_0 + b_1 x + b_2 x^2 + ... + b_{Nb} x^{Nb}$$

Розкриваємо дужки та прирівнюємо коефіцієнти при однакових степенях *x* (починаємо з нижчих степенів)

$$\left.\begin{aligned}
&x^0 : e_0 b_0 = a_0, \\
&x^1 : e_0 b_1 + e_1 b_0 = a_1, \\
&x^2 : e_0 b_2 + e_1 b_1 + e_2 b_0 = a_2, \\
&x^3 : e_0 b_3 + e_1 b_2 + e_2 b_1 + e_3 b_0 = a_3, \\
&\text{..... ....} \\
&x^n : e_0 b_n + e_1 b_{n-1} + e_2 b_{n-2} + ... + e_n b_0 = a_n
\end{aligned}\right\}$$

Записуючи ми за умовчанням вважали, що $b_s\big|_{s>Nb} = 0$ та $a_s\big|_{s>Na} = 0$ як воно і є насправді якщо *n>Na* та (чи) *n>Nb.* Якщо прийняти до уваги цю обставину та розглядати як систему алгебраїчних рівнянь відносно коефіцієнтів a_s, $0 \le s \le n$, то можна записати такі узагальнюючі формули

$$\begin{cases} e_0 = \dfrac{a_0}{b_0}, \\ e_z = \dfrac{a_z - \sum_{s=0}^{z-1} e_s b_{z-s}}{b_0}, 1 \le z \le n \end{cases}$$

Увага! Зверніть увагу на те, що b_0 не має права бути нулем.

Формула реалізується в процедурі EDrf.

```
   procedure EDrf (A, B:coef; n:byte; var E:coef;
var Na, Nb,z,s: integer; Sum:real
        begin
           Na:=round(A[-1]); Nb:=round(B[-1]);
           for s:=Na+1 to n do A[s]:=0;
           for s:=Nb+1 to n do B[s]:=0;
           E[-1]:=n; E[0]:=A[0]/B[0];
           For z:=1 to n do
              begin
                Sum:=0;
                for s:=0 to z-1 do
                        Sum:=Sum+E[s]*B[z-s];
                E[z]:=(A[z]-Sum)/B[0]
              end
        end;
```

Алгебраїчне додавання степеневих рядів

$$E(x)=A(x)\pm B(x),$$

де

$A(x)= a_0 + a_1 x + a_2 x^2 + \ldots + a_{Na} x^{Na} + \ldots,$

$B(x)= b_0 + b_1 x + b_2 x^2 + \ldots + b_{Nb} x^{Nb} + \ldots,$

$E(x)= e_0 + e_1 x + e_2 x^2 + \ldots + e_n x^n + \ldots,$

Наведений запис потрібно розуміти так що нескінченний ряд A(x) обірвано на члени $a_{Na}\ x^{Na}$, ряд B(x) на $b_{Nb}\ x^{Nb}$, а ряд E(x) на $e_n x^n$.Чому може доріванювати n? Нам уявляється, що читачеві очевидно, що n має не перевищувати меншого серед Na та Nb (адже усі три ряди нескінченні, і коефіцієнти при відкинутих доданках в A(x) та B(x) зовсім не нулі,просто ми їх не маємо, вони для нас недоступні, а звідси- сума доданків, де хоча б один доданок відсутній- неправомірна. Для визначеності будемо вважати, що n дорівнює меншому серед Na та Nb („Щоб добро не пропадало!”). Враховуючи сказане, а також замінючи на

E(x)=A(x)+C*B(x),

де C- коєфіцієнт множник типу real, одержуемо пидпрограму AlSumSer.

```
Procedure AlSumSer(A,B:Coef; C:real; var E:Coef);
```

```
Var  Na,Nb,Ne, s: integer;
 begin
    Na:=round(A[-1]); Nb:=round(B[-1]);
      if Na<Nb then  Ne:=Na
                  else Ne:=Nb;
                    E[-1]:=Na;
          for  s:=0   to Ne  do
              E[s]:=A[s]+C*B[s]
 end;
```

Множення рядів. Треба сформувати початковий відрізок ряду E(x), отриманого в результаті перелічення рядів A(x) та B(x), представленихсвоїми початковими відрізками.

E(x)=A(x)*B(x),

де $A(x)= a_0 + a_1 x + a_2 x^2 + \ldots + a_{Na} x^{Na} + \ldots$,

$B(x)=b_0 + b_1 x + b_2 x^2 + \ldots + b_{Nb} x^{Nb} + \ldots$,

$E(x)=e_0 + e_1 x + e_2 x^2 + \ldots + e_n x^n + \ldots$,

позначення тут такі як, в 2.6. Так чому туь дорівнює n? Знову читаєм меншому серед *Na* та *Nb*.

```
procedure UmnSer (A,B:Coef; var E:Coef);
    var Na,Nb,Ne, z,s: integer;
  begin
    Na:=round (A[-1]); Nb:=round(B[-1]);
    if Na<Nb    then Ne:=Na
                  else Ne:=Nb;
    E[-1]:=Ne;
     for s:=0  to Na  do  E[s]:=0;
     for z:=0  to  Ne  do
         for s:=0  to  Ne  do
          if z+s <=Ne   then
             E[z+s]:=E[z+s]+A[z]+B[s]
  end;
```

Ділення рядів. Задача практично аналогічна розв’язаній в 2.5 з тією лише різницею, що n треба приймати меншим серед *Na* та *Nb* з наведених там міркувань(на більше у нас просто немає матеріалу).

Відповідно модифікується процедура Edrf отримаємо:

```
Procedure DelSer (A,B:Coef; var E:Coef);
        var   Na,Nb,Ne,z,s: integer; Sum:real;
        begin
              Na:=round(A[-1]);  Nb:=round(B[-1]);
               if  Na<Nb  then Ne:=Na
                           else  Ne:=Nb;
```

```
E[-1]:=Ne;  E[0]:=A[0]/B[0];
 for z:=1  to Ne  do
   begin
        Sum i=0;
         for  s:=0  to  z-1  do
              Sum:=Sum+E[s]*B[z-s];
          E[z]:=(A[z]-Sum)/B[0]
    end
end;
```

Обчислення значення полінома при комплексному значенні аргумента

Скористаемось тією ж таки схемою Хорнера щоб обчислити значення полінома(2,18) при комплексному значенні $x=R+jI$, де R,I- дійсна та уявна частина x, $j=\sqrt{-1}$ - уявна одиниця.

Один крок (виток) в циклі, що реалізує схему Хорнера при комплексному x можна представити так

$$\mathrm{Re}^H + j\,\mathrm{Im}^H = (\mathrm{Re}^c + j\,\mathrm{Im}^c)(R + jI) + a_s,$$

де, як уже згадувалось $R+jI$- значення x ,

$Re+jIm$-поточне (в процесі реалізації схеми Хорнера) значення полінома.

a_s-черговий коєфіцієнт полінома.

Припустимо, що при комплексному x і значення полінома має бути в загальному випадку-комплексним $Re+jIm$.

$\mathrm{Re}^c + j\,\mathrm{Im}^c$ –„старе”(отримане на попередньому кроці) значення полінома.

$\mathrm{Re}^H + j\,\mathrm{Im}^H$ –„ нове”(оновлене) його значення.Комплексну рівність(2,20) можна представити двома дійсними рівностями.

$$\mathrm{Re}^H = \mathrm{Re}^c \cdot R - \mathrm{Im}^c \cdot I + a_s,$$

$$\mathrm{Im}^H = \mathrm{Re}^c \cdot I + \mathrm{Im}^c\, R.$$

Співвідношення покладено в основу підпрограми HorComp.

```
procedure HorComp (A:Coef; R,I:real; var Re, Im:real);
     var s,n:integer;
   begin
        n:=round(A[-1]); Re:=A[n]; Im:=0;
         for  s:=n-1 downto  0  do
             begin
                 R1:=Re*R-ImI+A[s];
                 Im:=Re*I+Im*R;
                 Re:=R1
            end
     end;
```

Верхні індекси(‘H’ та ‘C’-нове, старе) в підпрограмі HorComp це фігурують, оскільки ми користуємося тією властивістю оператора присвоювання, що в

його правій частині використовується „старі”, а в лівій „нові” значення ідентифікаторів, що повторюються. От тільки довелось ввести нову (проміжну) змінну R1:real, щоб запам’ятовувати оновлене значення Re не втрачаючи при цьому його старе значення, необхідне для оновлення Im. Після того, як необхідність у зберіганні старого Re відпадає, йому повертається його значення, що тимчасово зберігалось в R1.

Лекція 3. Критерій Гурвиця. Визначення ступеня стійкості.

Переконувати читача у важливості визначення коренів полінома нема потреби. Якщо, наприклад, досліджуваний поліном характеристичний, то маючи його корені можна отримати безліч надзвичайно цінної інформації про відповідну динамічну систему: буде вона стійкою чи ні, і який у неї запас стійкості, і буде вона аперіодичною, чи коливною, та яку частоту (чи частоти) матиме її коливна (коливні) складова, і як швидко затухатимуть коливання і взагалі весь перехідний процес. І цей список можна продовжувати.

І ще одна надзвичайно характерна обставина: поліном *n*-го степеня має рівно *n* коренів (не більше і не менше!). І якщо уже їх шукати, то як правило, усі і відразу.

Визначення круга, що охоплює усі корені полінома. Щоб цілеспрямовано шукати корені полінома, бажано хоча б орієнтовно уявляти де вони можуть знаходитись. Таку «наводку» дають математики []. Вони стверджують, що корені полінома не виходять за межі круга з центром в початку координат комплексної площини і радіусом

$$R = 1 + \frac{b}{|a_n|},$$

де $b = max\,(|a_0|, |a_1|, |a_2|, \ldots, |a_{n-1}|)$.

Якщо чесно, то математики вказують на кільце, зовнішній радіус якого визначається формулою , а внутрішній дорівнює

$$r = \frac{1}{1 + \frac{c}{|a_0|}},$$

де $c = max\,(|a_1|, |a_2|, \ldots, |a_n|)$.

Ми в подальшому при пошуку коренів поліномів будемо ігнорувати формулу (дірку в бублику) в розрахунку на те, що деяке збільшення зони пошуку (за рахунок дірки) компенсується спрощенням алгоритму.

Формулу реалізує підпрограма Radius.

```
function Radius (A: Coef): real;
        var s,n:integer; b:real;
        begin
            RedPol (Eps,A);
            n:= round (A[-1]); B:=abs(A[0]);
```

```
    for s:=1 to n-1 do
      if abs(A[s])>b then B:=abs(A[s]);
    Radius:=1+B/abs(A[n])
end;
```

Сподіваємось, читач звернув увагу на загрозу, що виникає при $a_n = 0$ (формула). Щоб відвернути її ми і викликаємо на початку процедуру RedPol. До речі, аналогічне ускладнення виникло б, якби ми спробували задіяти формулу – при $a_0 = 0$. У цьому випадку формулу краще було б представити у вигляді

$$r = \frac{|a_0|}{|a_0| + c}$$

і тоді проблеми з обчисленням *r* не було б.

Але повернемось до формули . З неї випливає, що коли $|a_n|$ наближається до нуля, то *R* прямує у нескінченність – зона пошуку необмежено розширюється. В принципі, можна вважати, що будь-який поліном *n*-го степеня (фактичного!) має *n* скінченних коренів і нескінченну кількість коренів на нескінченності, але останні нас не цікавлять.

Визначення дійсних коренів полінома Принципово визначення дійсних коренів полінома практично нічим не відрізняється від розв`язання нелінійних рівнянь загального виду

$$f(x) = 0.$$

Коли $f(x)$ шляхом алгебраїчних перетворень може бути приведено до вигляду полінома, то тоді рівняння називають алгебраїчним, в усіх інших випадках воно вважається трансцендентним. Отже, корені полінома -- це окремий випадок розв`язку рівняння . Почнемо з розгляду деяких загальних підходів до визначення дійсних коренів довільного рівняння . Перше, що треба вияснити: а чи взагалі це рівняння має дійсні корені, і якщо воно їх має, то де, хоча б орієнтовно ці корені знаходяться. Якщо у дослідника немає якоїсь апріорної інформації про досліджувану функцію $f(x)$, то найпростішим способом отримати таку інформацію буде побудувати графік функції. В п. 1.4. ми розглядали алгоритм формування графіків. Точки перетину графіка з віссю *x* – це і будуть шукані корені. У програмі Graphics користувач задає діапазон значень *x* для побудови графіка функції $f(x)$ -- це параметри *Xn* та *Xk*. Послідовно звужуючи цей діапазон таким чином, щоб точка перетину графіка з віссю *X* увесь час залишались в межах діапазону, можна визначити шуканий корінь практично з будь-якою бажаною точністю.

Розглянутий алгоритм локалізації кореня функції $f(x)$ з психологічної точки зору викликає інстинктивний супротив: о, то ще треба малювати графіки! Але ж малювання графіків із автоматичним формуванням системи координат та масштабуванням бере на себе комп’ютер, і кожен графік формується практично миттєво. Але, зрозуміло, користувач повинен увесь час приймати активну участь у процесі: лівіше, правіше, трохи менше, трохи

більше … Декому це цікаво, інших – стомлює. Автоматизуємо цей процес. Будемо сканувати заданий (бажаний, підозрюваний, очікуваний) діапазон *Xn…Xk* з деяким кроком *h*. Мета сканування: виявити відрізок довжиною в крок, на кінцях якого досліджувана функція приймає значення різних знаків. Це так званий алгоритм Хорнера-Руффіні відділення дійсного кореня.

Для реалізації даного алгоритму користувач має задати *Xn, Xk* та *h* (або відповідно *m*, таке, що *h=(Xk-Xn)/m*). Як? Вкрай бажано мати хоч якусь апріорну інформацію про функцію $f(x)$, ну, хоча б графік її побудуйте (заставте комп`ютер це зробити!). Не обов'язково вибирати *h* рівним або меншим допустимої похибки визначення кореня. Виконувати мільйони кроків сканування може «не сподобатися» навіть комп'ютеру. Можливо, варто спочатку «пройтись» вздовж діапазону з відносно великим кроком, знайшовши підозрілу ділянку – повторити її сканування з відповідно меншим кроком і так стільки разів, скільки будете вважати за потрібне.

Реалізуємо у вигляді підпрограми *HorRuf* однократне сканування діапазону *Xn…Xk* з метою відділення дійсного кореня функції.

```
function Fx(x: real): real;
```

Отже:

```
procedure HorRuf (m: word; var Xn, Xk:real; var F: boolean);
    var z,s:integer;
        h:real;
    begin
        h:= (Xk-Xn)/m; F:= false; s:=1;
        if Fx(Xn)>0 then z:=1
                    else z:=-1;
        repeat
            if Fx(s*h)*z<0 then F:=true
                           else inc(s)
        until F or (s>m):
        if F then
            begin
                Xk:=s*h; Xn:=Xk-h
            end
    end;
```

Тут *Xn* та *Xk* – початкове та кінцеве значення для початкового і водночас для кінцевого відрізка відділення (якщо такий буде виявлено в процесі сканування). *m* – число кроків сканування. *F = true*, якщо сканування завершилось успіхом (виявлено відрізок шириною *h*, на кінцях якого функція Fx(*x*) приймає значення різних знаків – корінь відділено) і *F = false,* якщо сканування закінчилося безрезультатно, в останньому випадку *Xn* та *Xk* якими були такими і залишаться.

Якщо на досліджуваному відрізку функція має декілька не співпадаючих коренів, то розгляданий алгоритм має відділити крайній зліва корінь (у загальному випадку це може виявитися коренем непарної кратності). Корені парної кратності цей алгоритм «не ловить» (графік функції

в корені парної кратності дотикається до осі X не перетинаючи її, не змінюючи свого знака).

Щоб виділити (відділити) корінь у випадку близьких (але не кратних) коренів крок сканування має бути меншим відстані між близькими «сусідами» («розв'язуючи задачу на комп'ютері, відповідь треба знати наперед»).

В підпрограмі *HorRuf z* – знак функції $f(x)$ на лівому кінці досліджуваного інтервалу. Зміну знаку на черговому кроці часто рекомендують визначати за тим, що добуток значень функції на кінцях кроку (перед кроком та після нього) стає від'ємним. Але ж множити на ± 1 простіше, ніж на значення функції перед кроком.

І ще одне, ми не відслідковуємо ситуацію, коли $f(x)$ перетворюється на нуль, або (в більш загальному випадку) стає за модулем менше ε, тобто фактично ігноруємо випадок, коли шуканий корінь попадається «на гарячому». Чому так? Якщо чесно, то ми не знаємо, як вибрати ε. Якщо, наприклад, досліджувану функцію помножити (поділити) на мільйон, то як, на вашу думку, її корені від цього зміняться? А значення функції? А ще ж до того – похибки в обчисленні функції. Ви їх знаєте? Ми – ні.

Сказаним відносно відділення коренів функцій (і поліномів зокрема) обмежимось.

Наступний етап – уточнення кореня, тобто його визначення з заданим рівнем точності (з похибкою, що не перевищує ε -- заданої малої величини). Не плутати дане ε з ε, що визначає відхилення значення функції від нуля! Існує ряд алгоритмів уточнення дійсного кореня. Їх можна поділити на такі, що сходяться умовно і безумовно. Нас будуть цікавити в першу чергу ті, що сходяться безумовно.

Ми уже згадували варіант використання для уточнення кореня метода сканування. Оформимо цю ідею у вигляді підпрограми

```
function RootHR (m: word; Xn, Xk, Eps: real): real;
        var F: boolean;
        begin
                repeat
                        HorRuf(m, Xn, Xk, F)
                until Xk- Xn<2*Eps;
                RootHR:=( Xn+ Xk)/2
        end;
```

Тут параметр F в підпрограмі HorRuf виявився незадіяним. Справа в тому, що функцію RootHR є сенс викликати лише після того, як корінь гарантовано відділений, отже F при чергових викликах підпрограми HorRuf гарантовано буде повертатись з F:=true, а тому контролювати його нема потреби.

Операторна частина підпрограми RootHR завершується оператором

```
RootHR:=( Xn+ Xk)/2,
```

коли ми остаточний результат (уточнене значення кореня) беремо як середину відрізка відділення.

Це ідея тільки більш послідовно реалізується у методі уточнення, який так і називається «метод половинного ділення» (дихотомії, бісекції, метод Больцано). Отже, виходимо з того, що корінь відділено (функція на кінцях відрізка має протилежні знаки). Ділимо відрізок пополам, обчислюємо значення функції в центральній точці і порівнюємо її знак, скажімо, зі знаком її на лівому кінці відрізка. Якщо знаки однакові, то лівий кінець відрізка переносимо в центр, інакше – в центр переноситься правий кінець. В результаті ширина відрізка зменшується вдвічі. Повторюючи цей процес необхідну кількість разів можна звузити відрізок настільки, що його ширина не буде перевищувати 2ε, отже центр даного відрізка буде відрізнятись не більше ніж на ε від справжнього значення кореня.

```
function Dych (Xn, Xk, Eps: real): real;
    var x: real;
        Zn: integer;
    begin
        if Fx(Xn)>0 then Zn:= 1
                    else Zn:= -1;
        repeat
            x:=( Xn+ Xk)/2;
            if Fx(x)*Zn>0 then Xn:=x
                          else Xk:= x
        until Xk- Xn<=2*Eps;
        Dych:= (Xn+ Xk)/2
    end;
```

Другим методом, що безумовно сходиться, ми розглянемо метод хорд. Тут на відміну від методу половинного ділення чергова точка x – це точка перетину з віссю Хорди, що з'єднує значення функції на кінцях відрізка відділення кореня. Абсциса точки x легко визначається з умови подібності трикутників.

$$\frac{Fn}{x - Xn} = \frac{-Fk}{Xk - x},$$

де Fn, Fk – значення функції в точках Xn, Xk.

З визначається x

$$x = \frac{Fn \cdot Xk - Fk \cdot Xn}{Fn - Fk}.$$

```
function Cord (Xn, Xk, Eps: real): real;
    var X, F, Fn, Fk: real;
        Zn: integer;
    begin
```

```
        Fn:= Fx(Xn); Fk:= Fx(Xk);
        if Fn>0 then Zn:= 1
                else Zn:= -1;
        repeat
                X:= (Fn*Xk-Fk*Xn)/(Fn-Fk);
                F:= Fx(x);
                if F*Zn>0
                        then
                                begin
                                        Xn:= X; Fn:= F
                                end
                        else
                                begin
                                        Xk:= X; Fk:= F
                                end
        until Xk- Xn<2*Eps;
        Cord:= (Xn+Xk)/2
end;
```

Тут останнім оператором значення Cord визначається шляхом половинного ділення, хоча, звичайно, можна було б скористатись і формулою . Але – ми обіцяли точність не гірше ε -- маєте!

Коли б йшлося про корінь полінома, то просто замість (або в складі) функції Fx(*x*) фігурувала б функція HorReal (A,*x*), де A:Coef – масив з інформацією про досліджуваний поліном.

Якщо функція F(*x*) на ділянці *Xn…Xk* неперервна (а поліном є неперервною функцією на усій числовій осі *x*), то, власне, особливої потреби в розгляді інших алгоритмів уточнення кореня функції і немає. За виграш у кілька відсотків у трудомісткості інші методи (як, наприклад, метод дотичних (Ньютона), метод параболічної інтерполяції і т. д. і т. п.) ми можемо отримати цілком реальний головний біль у вигляді процесу, що розходиться. То чи варто?

Досі ми розглядали алгоритми визначення дійсних коренів функції $f(x)$ - і полінома зокрема. Тоді як для полінома знати його дійсні корені може виявитися абсолютно недостатнім. Більш того, як показує досвід, найбільший інтерес для широкого кола проблем представляють саме комплексні корені поліномів.

Пошук комплексного кореня полінома. У точці комплексної площини, яка є коренем полінома значення полінома має дорівнювати 0. Це означає, що дійсна і уявна частини значення полінома мають одночасно перетворюватись на нулі. Цю ж умову можна сформулювати і так: модуль значення полінома стає рівним нулю. Так може варто «прогулятись» в межах визначеного круга та пошукати точки, де б модуль значення полінома приймав глобально найменше значення. Абсолютний мінімум для модуля – нуль.

Зробимо так. Опишемо навколо круга квадрат, щоб простіше було організувати сканування. Отриманий таким чином квадрат покриємо квадратною сіткою з кроком $\frac{R}{m}$, де R – вищезгаданий радіус круга (він же зараз є половиною сторони квадрата), m – деяке наперед задане ціле число (кількість кроків на половину сторони квадрата). Перебираємо послідовно вузли сітки. Обчислюємо в кожному з них значення *Re+jIm* полінома, яке тут же перетворюємо в модуль $\left|\sqrt{Re^2+Im^2}\right|$ або квадрат модуля $(\ Re^2+Im^2\ )$ та фіксуємо вузол, де цей модуль (квадрат модуля) приймає найменше значення. Якби цей модуль виявився рівним нулю, то даний вузол якраз і був би шуканим коренем полінома. Знайдений вузол приймається за центр зменшеного квадрата розмірами $2h\times 2h$ (по кроку вліво, вправо, вверх та вниз від знайденого вузла-центра). Зменшений квадрат скануємо з кроком в *m* раз меншим від попереднього і так доти, поки не буде завершено сканування з кроком, що не перевищує ε (допустимої похибки визначення коренів). Знайдений «оптимальний» вузол будемо приймати за корінь полінома.

```
procedure KorPolSc (A: Coef; m: integer; Eps: real; var Rk,Ik: real);
    var h, R, I, Rc, Ic, Re, Im, M2, MinMod, Rad: real;
    begin
        Rad:=Radius(A); Rc:=0; Ic:=0;
        Rk:=0; Ik:=0 MinMod:=1e25;
        h:=Rad/m;
        repeat
            for z:=-m to m do
            begin
                I:=Ic+Z*h;
                for S:=-m to m do
                begin
                    R:=Rc+s*h;
                    HorComp(A, R, I, Re, Im);
                    M2:=sqr(Re)+sqr(Im);
                    if M2<MinMod then
                    begin
                        Rk:=R; Ik:=I;
                        MinMod:=M2
                    end
                end
            end
        if h>Eps then
        begin
            Rc:=Rk; Ic:=Ik; h:=h/m
        end
        until h<Eps
```

end;

Тут *Rc, Ic* – координати центра досліджуваного квадрата; *R, I* – координати вузла; *Rk, Ik* – координати поточного «кращого» вузла;
MinMod – поточне визначення кращого (мінімального) модуля; M2 – квадрат модуля значення полінома в поточному вузлі. Чому квадрат модуля, а не сам модуль? Та лише тому, щоб не обчислювати корінь квадратний. Адже нас цікавить не стільки сам найменший модуль, скільки місце, де він знаходиться. Так мінімум квадрата модуля знаходиться там же, де і мінімум самого модуля. «Навіщо платити більше?...»

Враховуючи високу швидкодію сучасних комп'ютерів розглянутий варіант не варто відкидати «з порога» як надто примітивний та надто трудомісткий (що було абсолютно справедливо у «докомп'ютерний» час). Надійність алгоритму підвищується при збільшенні *m*, але – за рахунок збільшення обсягу обчислень. Мабуть не варто брати *m* надто великим (при *m*=1000 кількість вузлів буде більше 4 мільйонів!). Конкретний вибір *m* – за користувачем. Це настроювальний параметр. Не варто позбавляти користувача необхідності хоч трохи думати.

Далі ми збираємось розглянути деякі нетрадиційні алгоритми визначення коренів полінома (як дійсних так і комплексних), що відзначаються абсолютною надійністю (сходимістю), яка (як і всяка перевага) досягається певним збільшенням обсягу обчислень.

Критерій Гурвиця дає відповідь на одне просте запитання – чи має досліджуваний поліном корені в правій половині комплексної площини. Результат на перший вигляд більше ніж скромний. Але не будемо поспішати з висновками.

Розглянемо для прикладу поліном степеня n=6

$$A(x) = a_6x^6 + a_5x^5 + a_4x^4 + a_3x^3 + a_2x^2 + a_1x + a_0$$

З коефіцієнтів цього полінома формуємо так званий визначник Гурвиця:

$$\Delta = \begin{Vmatrix} a_5 & a_3 & a_1 & 0 & 0 & 0 \\ a_6 & a_4 & a_2 & a_0 & 0 & 0 \\ 0 & a_5 & a_3 & a_1 & 0 & 0 \\ 0 & a_6 & a_4 & a_2 & a_0 & 0 \\ 0 & 0 & a_5 & a_3 & a_1 & 0 \\ 0 & 0 & a_6 & a_4 & a_2 & a_0 \end{Vmatrix}$$

Алгоритм заповнення визначника Гурвиця можна сформулювати так. Але спершу поліном треба підготувати.

Перше. Досліджуваний поліном записується в порядку зменшення степенів *x*, при чому в запису мають фігурувати усі доданки (зі степенями *x* від *n* до 0) в тому числі і доданки з нульовими коефіцієнтами.

Друге. Старший коефіцієнт a_n має відрізнятись від нуля (редагування полінома!).

Третє. Старший коефіцієнт a_n має бути додатнім. Якщо це не так, треба поміняти знаки усіх коефіцієнтів полінома на протилежні. Поліном від цього зміниться, але не зміняться його корені, а для нас зараз саме це є суттєвим.

Строго кажучи, виконання третьої позиції не є обов`язковим, але це значно спрощує формулювання критерія, тому ми будемо слідкувати за виконанням і цієї позиції.

А тепер – алгоритм формування визначника Гурвиця. Вздовж головної діагоналі визначника записуємо послідовно (зліва направо) коефіцієнти полінома, починаючи з передстаршого (з a_{n-1}, у нашому випадку з a_5) – закінчуючи a_0. Вниз (по стовпчиках) від кожного діагонального елемента послідовно вписуємо коефіцієнти, що стоять у поліномі лівіше даного діагонального, а вверх – коефіцієнти, що стоять правіше діагонального. Якщо коефіцієнти «закінчуються» -- пишемо нулі. Формула демонструє описану схему.

Можна звернути увагу на повторюваність рядків (через один зі зсувом на одну позицію вправо, якщо йти зверху вниз).

І, нарешті, формулювання критерія Гурвиця. Поліном, у якого $a_n>0$, не має коренів у правій півплощині (з додатною дійсною частиною) тоді і тільки тоді, коли всі діагональні мінори визначника Гурвиця будуть строго додатніми.

Формально, для полінома умови критерія Гурвиця виглядають так:

$$\begin{cases} a_n > 0. \\ \Delta_1 = \|a_5\| > 0, \\ \Delta_2 = \begin{Vmatrix} a_5 & a_3 \\ a_6 & a_4 \end{Vmatrix} > 0, \\ \Delta_3 = \begin{Vmatrix} a_5 & a_3 & a_1 \\ a_6 & a_4 & a_2 \\ 0 & a_5 & a_3 \end{Vmatrix} > 0, \\ \ldots \\ \Delta_6 = \Delta_5 \cdot a_0 > 0. \end{cases}$$

Ну, от – скаже читач – невистачало ще обчислювати визначники! Загальний комп'ютерний алгоритм обчислення визначників ми розглянемо нижче, а поки що, постараємося скористатися специфікою визначника Гурвиця, щоб спростити перевірку умов .

Для цього шляхом еквівалентних перетворень (які не змінюють значення визначника та його діагональних мінорів) приведемо визначник Гурвиця до трикутного вигляду, коли всі його елементи, розташовані нижче (і лівіше) від головної діагоналі, перетворяться на нулі.

Почнемо з того, що спробуємо онулити елемент a_6 у другому рядку визначника. Для цього з другого рядка поелементно вираховуємо перший

рядок, помножений на деякий коефіцієнт R_6, результат занесемо в другий рядок. Коефіцієнт R_6 підбираємо таким чином, щоб перший елемент оновленого другого рядка перетворився б на нуль, тобто: $a_6 - R_6 a_5 = 0$,
звідки

$$R_6 = \frac{a_6}{a_5}.$$

В результаті нові значення елементів другого рядка стануть такими

$$\begin{cases} a_6^* = 0, \\ a_4^* = a_4 - R_6 a_3, \\ a_2^* = a_2 - R_6 a_1, \\ a_0^* = a_0 - R_6 0 = a_0. \end{cases}$$

А тепер приймемо до уваги, що третій рядок повторює (зі зсувом на одну позицію вправо) 1-й рядок, а 4-й дублює 2-й (з таким же зсувом). Отже якщо з 4-го рядка вирахувати третій помножений на R_6 , і записати результат в 4-ий рядок, то, якщо не рахувати першого і останнього елементів (які як були так і залишаються нулями), елементи четвертого рядка після оновлення приймуть значення, що визначаються тими ж таки формулами – додатково нічого перераховувати нема потреби. Те ж саме можна сказати про 6-й рядок.

В результаті визначник приймає форму

$$\Delta = \begin{Vmatrix} a_5 & a_3 & a_1 & 0 & 0 & 0 \\ 0 & a_4^* & a_2^* & a_0 & 0 & 0 \\ 0 & a_5 & a_3 & a_1 & 0 & 0 \\ 0 & 0 & a_4^* & a_2^* & a_0 & 0 \\ 0 & 0 & a_5 & a_3 & a_1 & 0 \\ 0 & 0 & 0 & a_4^* & a_2^* & a_0 \end{Vmatrix}$$

Тепер переходимо до онулення a_5 в третьому рядку . Для цього поставимо вимогу, щоб $a_5 - a_4^* \cdot R_5 = 0$
тобто, щоб

$$R_5 = \frac{a_5}{a_4^*}$$

і

$$\begin{cases} a_5^* = 0, \\ a_3^* = a_3 - R_5 a_2^*, \\ a_1^* = a_1 - R_5 a_0. \end{cases}$$

5-ий рядок повторює 3-ій (зі зсувом). В результаті перетворень , визначник трансформується в

$$\Delta = \begin{Vmatrix} a_5 & a_3 & a_1 & 0 & 0 & 0 \\ 0 & a_4* & a_2* & a_0 & 0 & 0 \\ 0 & 0 & a_3* & a_1* & 0 & 0 \\ 0 & 0 & a_4* & a_2* & a_0 & 0 \\ 0 & 0 & 0 & a_3* & a_1* & 0 \\ 0 & 0 & 0 & a_4* & a_2* & a_0 \end{Vmatrix}$$

На черзі a_4* в четвертому рядку :

$$R_4 = \frac{a_4*}{a_3*}$$

та

$$\begin{cases} a_4** = 0, \\ a_2** = a_2* - R_4 a_1*. \end{cases}$$

Аналогічно 4-му рядку перетворюється 6-ий рядок. Визначник набуває вигляду

$$\Delta = \begin{Vmatrix} a_5 & a_3 & a_1 & 0 & 0 & 0 \\ 0 & a_4* & a_2* & a_0 & 0 & 0 \\ 0 & 0 & a_3* & a_1* & 0 & 0 \\ 0 & 0 & 0 & a_2** & a_0 & 0 \\ 0 & 0 & 0 & a_3* & a_1* & 0 \\ 0 & 0 & 0 & 0 & a_2** & a_0 \end{Vmatrix}$$

Онулюємо a_3* в 5-му рядку

$$R_3 = \frac{a_3*}{a_2**},$$

$$\begin{cases} a_3** = 0, \\ a_1** = a_1* - R_3 a_0. \end{cases}$$

Отже, перетворюється на

$$\Delta = \begin{Vmatrix} a_5 & a_3 & a_1 & 0 & 0 & 0 \\ 0 & a_4* & a_2* & a_0 & 0 & 0 \\ 0 & 0 & a_3* & a_1* & 0 & 0 \\ 0 & 0 & 0 & a_2** & a_0 & 0 \\ 0 & 0 & 0 & 0 & a_1** & 0 \\ 0 & 0 & 0 & 0 & a_2** & a_0 \end{Vmatrix}$$

Останній поки що не онулений елемент – це a_2** в 6-му рядку . Для цього

$$R_2 = \frac{a_2**}{a_1**}$$

та $a_0 = a_0 - R_2 0 = a_0$.

6-ий рядок в результаті такого перетворення практично не змінюється, отже, перетворення та є фактично зайвими.

Обчислення значень діагональних мінорів визначника Гурвиця звелось до перемноження діагональних елементів визначника . Тим самим

умови Гурвиця перетворюються на вимоги, щоб a_5>0, a_4*>0, a_3*>0, a_2**>0, a_1**>0, a_0>0 (усі елементи головної діагоналі визначника мають бути строго додатніми).

Щойно розглянутий варіант реалізації критерія Гурвиця у вигляді послідовності формул – – це фактично алгоритм Рауса. З цієї причини розглянутий критерій часто називають критерієм Гурвиця-Рауса. Модифікація критерія Гурвиця у формі Рауса добра ще й тим, що її можна реалізувати оперуючи лише одновимірним масивом A:Coef коефіцієнтів полінома. Так, для полінома у початковому стані масив A має вигляд

	-1	0	1	2	3	4	5	6
A	6	a_0	a_1	a_2	a_3	a_4	a_5	a_6

Виконуємо обчислення –

$$R_6 = \frac{a_6}{a_5}, \qquad a_4* = a_4 - R_6 a_3, \qquad a_2* = a_2 - R_6 a_1.$$

Масив A набуває вигляд

	-1	0	1	2	3	4	5	6
A*	6	a_0	a_1	a_2*	a_3	a_4*	a_5	a_6

Тоді виконуємо обчислення –

$$R_5 = \frac{a_5}{a_4*}, \qquad a_3* = a_3 - R_5 a_2*, \qquad a_1* = a_1 - R_5 a_0.$$

Маємо

	-1	0	1	2	3	4	5	6
A**	6	a_0	a_1*	a_2*	a_3*	a_4*	a_5	a_6

Перетворення –

$$R_4 = \frac{a_4*}{a_3*}, \qquad a_2** = a_2* - R_4 a_1*$$

та

	-1	0	1	2	3	4	5	6
A***	6	a_0	a_1*	a_2**	a_3*	a_4*	a_5	a_6

І нарешті перетворення –

$$R_3 = \frac{a_3*}{a_2**}, \quad a_1** = a_1* - R_3 a_0$$

і

	-1	0	1	2	3	4	5	6
A****	6	a_0	a_1**	a_2**	a_3*	a_4*	a_5	a_6

Елементи A****, як легко бачити повторюють головну діагональ визначника . І перевірку виконання умов Гурвиця (знаків діагональних елементів визначника , а, отже, і елементів A****) варто виконувати в міру того, як послідовно приймають свої остаточні значення (зафіксовані в A****) елементи a_5, a_4*, a_3*, a_2**, a_1** та a_0. З наведеного прикладу видно, що коефіцієнти a_n, a_{n-1} та a_0 в процесі реалізації алгоритму Рауса, не змінюються, і стосовно до них умови Гурвиця зводяться до

$a_n > 0$, $\Delta_1 = a_{n-1} > 0$, $\Delta_n = \Delta_{n-1} \cdot a_0 > 0$,

себто умови ці – просто, щоб коефіцієнти a_n, a_{n-1} та a_0 були строго додатніми. А тепер поглянемо на поліном виду (в загальному випадку n-го степеня) дещо з іншого боку. Його ж можна представити й так

$$A(x) = a_n(x - x_1)(x - x_2)...(x - x_n),$$

де x_1, x_2, …, x_n -- корені цього полінома.

Критерій Гурвиця-Гауса перевіряє, чи усі корені полінома мають від'ємні дійсні частини. Припустимо, що - так. Тоді дійсний корінь, скажімо, x_1 має бути від'ємним і відповідний йому множник може бути представлений так

$(x + |x_1|)$.

Якщо маємо пару спряжених комплексних коренів, наприклад,

$x_{2,3} = \alpha \pm j\beta$,

то відповідна цій парі пара множників у представляється таким чином

$(x - \alpha - j\beta)(x - \alpha + j\beta) = (x + |\alpha| - j\beta)(x + |\alpha| + j\beta) = (x + |\alpha|)^2 + \beta^2 = x^2 + 2|\alpha|x + \alpha^2 + \beta^2$.

Це квадратний тричлен з додатніми коефіцієнтами! Добуток будь-якої кількості множників виду $(x + |x_1|)$ та $(x^2 + 2|\alpha|x + \alpha^2 + \beta^2)$ не може дати полінома, у якого хоча б один коефіцієнт був від'ємний, чи бодай рівним нулю.

Отже, об`єднуючи цей висновок з умовами можна сформулювати необхідну (а при $n \le 2$ і достатню) умову відсутності коренів з додатною дійсною частиною: усі коефіцієнти мають бути обов'язково додатніми. Сподіваємось, що сказаного достатньо, щоб зрозуміти механізм роботи підпрограми-функції Hurvic.

```
function Hurvic (A:Coef):integer;
    const Eps=1e-9;
    var z,s,Hurv,n:integer;
        R:real;
    begin
        Hurv:=1: n:=round(A[-1]);
        if n<0 then
            while (abs(A[n])<Eps) and (n>0) do dec(n);
        if n>0 then
```

```
begin
        if A[n]<0 then
        for z:=0 to n do A[z]:=-A[z];
        for z:=0 to n-1 do
        if A[z]<=Eps then Hurv:=-1;
        if (Hurv=1) and (n>2) then
        begin
                z:=n;
                while (z>2) and (Hurv=1) do
                begin
                        R:=A[z]/A[z-1]; s:=z-2;
                        A[s]:=A[s]-R*A[s-1];
                        if A[s]<Eps then Hurv:=-1
                                        else
                                         begin
                                           dec(s,2);
                                           while s>0 do
                                           A[s]:=A[s]-R*A[s-1]
                                         end;
                                    dec(z)
                end;
        end
    end;
  Hurvic:=Hurv
  end;
```

Декілька зауважень.

Чому функція Hurvic повертає значення типу integer, а не, скажімо, типу Boolean? Та просто цей алгоритм реалізації відпрацьовувався авторами на алгоритмічних мовах, де логічних змінних не було. Отже, і довелось домовлятися, що відповідь +1 означає (коренів справа від уявної осі нема), та -1 в інших випадках. А уже потім не хотілося псувати уже напрацьовані алгоритмічні продукти. «Історично так склалось…»

Константа Eps (відповідає ε) – навіщо?

Нам потрібна перевірка отримуваних елементів головної діагоналі визначника Гурвиця на строгу достатність.Але ж коефіцієнти полінома, переданого нам на дослідження, відомі з похибкою, крім того ми самі ведемо обчислення з похибками, отже, елементи, що ми маємо перевіряти на строгу додатність, відомі з похибкою. Так от Eps – це наш страховий поліс при винесенні «вердикту» відносно строгої додатності. Якщо модуль елемента, що перевіряється, менше Eps, то ми віддаємо перевагу негативному висновку. Ну а тепер, чому Eps=1e-9? При обчисленнях з одинарною точністю ми отримуємо число с 11-12 значущими цифрами. Якщо припустити, що ми втрачаємо 2-3 значущих цифри за рахунок імовірних похибок, то це якраз і буде десь біля 1e-9.

Якщо у читача є підстава думати інакше – скоригуйте Eps.

Операторна частина функції Hurvic починається з Hurv:=1. Тут Hurv – проміжне значення відповіді (щоб уникнути рекурсії). Початок – на оптимістичній ноті (Hurv=1 – отже, підстав вважати, що «праві» корені є – у нас немає – адже ми ще нічого не перевіряли).

n – степінь полінома.

Якщо n=0, то поліном 0-го степеня має 0 коренів, при відсутності коренів, остерігатись, що якийсь з них «правий» -- виключається.

Якщо ж n>0, то в циклі while визначаємо фактичний степінь полінома (опускаємо старші коефіцієнти, що не перевищують за модулем Eps), тобто редагуємо поліном. Якщо після редагування n залишилось більшим нуля, то перевіряємо знак A[n].Якщо A[n]<0, то змінюємо знаки всіх коефіцієнтів на протилежні.

Далі в циклі

for z:=0 to n do …

перевіряємо виконання необхідної умови. Коли бодай один з коефіцієнтів такої перевірки не витримує, то Hurv=-1. Якщо до цього часу збою не відбулось, і крім того степінь полінома більше двох, то приступаємо до реалізації основної частини алгоритму Рауса.

Починаємо з z:=n;

У циклі while (z>2) and (Hurv=1) do… обчислюємо R (це послідовно R_6, R_5, …, R_3). Позначаємо s:=z-2; Модифікуємо A[s]. Коли виявляється що модифікований A[s] (це черговий діагональний елемент трикутного визначника Гурвиця) від'ємний – процес зупиняється. (Hurv:=-1), інакше модифікуються з кроком – 2 коефіцієнти при даному R. z зменшується на одиницю (dec(z)) і тіло циклу while (z>2) and (Hurv=1) do… повторюється черговий раз, поки не буде вичерпана умова z>2 and (Hurv=1).

А тепер залишається лише перенести Hurv в Hurvic. В підручниках, де розглядаються алгоритми визначення коренів полінома, критерій Гурвиця, як правило не згадується. Критерій задуманий як засіб оцінки стійкості лінійної системи за її характеристичним рівнянням (відсутність правих коренів у характеристичного полінома якраз і гарантує стійкість). Деякі автори, поліноми, для яких виконуються умови критерія Гурвиця, називають стійкими (чи нестійкими). Хоча це, звичайно ж, умовно. Сам по собі поліном не може бути стійким чи не стійким. Але коли прийняти таку термінологію, то відносно полінома можна визначати не лише його стійкість, але і «ступінь стійкості».Щоб визначати ступінь стійкості нам спочатку необхідно розглянути допоміжну задачу, а саме зсув уявної осі на комплексній площині значень аргумента полінома.

Замінимо в поліномі A(x) змінну $\boldsymbol{x}$ на x_1-L , де x_1 – нова змінна, L – константа типу real (може бути як додатною, так і від'ємною).В результаті заміни отримаємо поліном

$$B(x_1) = A(x)\ \big|_{x=x_1-L}.$$

Очевидно, що поліном $B(x_1)$ матиме той же степінь, що й поліном A(x).
Виконаємо в ще одну заміну, а саме $x_1 = x + L$, матимемо:

$$B(x+L)=A(x)$$

або в розгорнутому вигляді

$$b_0+b_1(X+L)+b_2(X+L)^2+...+b_n(X+L)^n=A(x)$$

Покладаючи, що $x=-L$ (коли $x+L=0$) з одержуємо $b_0=A(x)\big|_{x=-L}$.

Диференціюємо за x

$$b_1+2b_2(X+L)+3b_3(X+L)^2+...+nb_n(X+L)^{n-1}=\frac{dA(x)}{dx},$$

звідки при $x=-L$ маємо

$$b_1=\left.\frac{dA(x)}{dx}\right|_{x=-L}$$

Тепер уже диференціюємо , звідки маємо

$$2b_2+3\cdot 2\cdot b_3(x+L)+...+n(n-1)b_n(x+L)^{n-2}=\frac{d^2A(x)}{dx^2}$$

та

$$b_2=\frac{1}{2!}\left.\frac{d^2A(x)}{dx^2}\right|_{x=-L}$$

Продовжуючи цей процес отримаємо в загальному вигляді

$$b_s=\frac{1}{s!}\left.\frac{d^sA(x)}{dx^s}\right|_{x=-L},0\le s\le n$$

При практичній реалізації формули верхню межу для b можна знизити до $n-1$, якщо враховувати, що $b_n=a_n$ (очевидно?!).

Формулу покладено в основу підпрограми ZamP, що наводиться нижче.

```
procedure ZamP (A:Coef; L:real; var B:Coef);
        var s,n:integer; F:real;
        begin
                n:=round(A[-1]); B[-1]:=n;
                B[n]:=A[n]; F:=1;
                for s:=0 to n-1 do
                begin
                        B[s]:=HorReal(A,-L)/F;
                        DifPol(A,A); F:=F*(s+1)
                end
         end;
```

Тут варто зробити одне зауваження. Змінна F – це факторіал (s! в формулі). За визначенням факторіал – ціле число. А ми описали F, як змінну типу real. Чому? Справа в тому, що

0!=1, 1!=1, 2!=2, 3!=6, 4!=24, 5!=120, 6!=720, 7!=5040, 8!=40320, 9!=362880… Коли б ми описали F:integer, то уже 8! вийшло б за допустимі межі типу integer(-32768…32767), а 9! вийшло б за допустимі межі типу word (0…65535). Компілятор Турбо Паскаля не сигналізує про вихід змінних цих типів за допустимі межі, що може загрожувати повною дезорганізацією обчислень. Отже віднесення F до типу real є меншим злом. Ми зараз не розглядаємо використання таких «екзотичних» типів як Longint, Double чи Extended.

Залучення процедурою ZamP підпрограм HorReal та DifPol (п. 2.10, п. 2.9) може дещо ускладнювати її реалізацію, тому враховуючи відносну простоту алгоритмів цих підпрограм, «розчинимо» їх в результуючій підпрограмі.

```
procedure ZamPPL (A:Coef; L:real; var B:Coef);
        var z,s,n:integer; F,Sum:real;
        begin
            n:=round(A[-1]); B[-1]:=n;
            B[n]:=A[n]; F:=1;
            for s:=0 to n-1 do
            begin
                Sum:=A[n-s];
                for z:=n-s-1 downto 0 do
                  Sum:=-Sum*L+A[z];
                B[s]:=Sum/F;
                for z:=1 to n-s do
                    A[z-1]:=A[z]*z;
                F:=F*(s+1)
            end
        end;
```

Саме цим варіантом підпрограми ми збираємось користуватися надалі.

Лекція 4. Розв’язання систем лінійних алгебраїчних рівнянь за схемою Гауса

Пусть имеем систему уравнений

$$\begin{cases} a_{1,1}x_1 + a_{1,2}x_2 + a_{1,3}x_3 + \ldots + a_{1,n}x_n = a_{1,n+1}, \\ a_{2,1}x_1 + a_{2,2}x_2 + a_{2,3}x_3 + \ldots + a_{2,n}x_n = a_{2,n+1}, \\ a_{3,1}x_1 + a_{3,2}x_2 + a_{3,3}x_3 + \ldots + a_{3,n}x_n = a_{3,n+1}, \\ \qquad\qquad\qquad \ldots \\ a_{n,1}x_1 + a_{n,2}x_2 + a_{n,3}x_3 + \ldots + a_{n,n}x_n = a_{n,n+1}. \end{cases}$$

Здесь x_1, x_2, x_3, …, x_n – неизвестные, подлежащие определению: $a_{1,1}$, $a_{1,2}$, …, $a_{n,n}$, $a_{n,n+1}$ – коэффициенты.

Предполагается, что количество уравнений равно числу неизвестных. Ограничимся рассмотрением только такой ситуации, хотя математики в разделе линейной алгебры рассматривают еще много иных вариантов.

Задача заключается в нахождении такого комплекта (массива) значений x_1, x_2, x_3, …, x_n, который бы удовлетворял каждое из уравнений системы. Если такое удастся найти, то это будет решение системы . Хотя не исключено, что система решения не имеет. В таких случаях говорят, что какие-то из уравнений (или даже все!) – несовместимы.

К подобной перспективе не следует относиться как к трагедии. «На нет – и суда нет!» – это нормально. Для иллюстрации представим себе систему из 2-х уравнений:

$$\begin{cases} a_{1,1}x_1 + a_{1,2}x_2 = a_{1,3}, \\ a_{2,1}x_1 + a_{2,2}x_2 = a_{2,3}. \end{cases}$$

В системе координат $x_1 0 x_2$ каждое из уравнений – уравнение прямой линии. Итак, две прямые на плоскости. Если они пересекаются – координаты точки пересечения и образуют решение системы , значит, система оказалась совместимой. Но ведь прямые могут оказаться параллельными. А таковые, как известно, если и пересекаются, то не ближе, чем на бесконечности. А решение на бесконечности, что оно есть – что его нет. Вот вам и несовместимость, и отсутствие решения.

А то – не легче – прямые могут и совпадать. Тогда любую точку любой (тождественных) прямой можно рассматривать как точку их пересечения то бишь решение системы. Это сколько же решений! И мало плохо, и много не легче. Договоримся (да простят нас математики) «сверхизобилие» приравнивать «отсутствию» – и там и там единого (единого!) решения нет. При решении практических задач такой подход чаще всего себя оправдывает, а более тонкий анализ выходит за рамки нашей книги, – «вам надо – копайте глубже».

Методов решения системы типа существует множество. Ограничимся методом последовательного исключения неизвестных – так называемый алгоритм Гаусса.

Начнем с того, что выразим x_1 из первого уравнения через остальные неизвестные (x_2, x_3, …, x_n) и правую часть $a_{1,n+1}$, естественно. А теперь это «значение» x_1 подставим во 2-е, 3-е, …, n-е уравнения (стоящие ниже 1-го). Во всех этих уравнениях после данной подстановки x_1 как неизвестная исчезнет (за счет модифицирования коэффициентов этих уравнений). А теперь из 2-го уравнения модифицированной системы определим x_2 (через x_3, x_4, …, x_n и модифицированный $a_{2,n+1}$) и подставим в 3-е, 4-е, …, n-е уравнения. Таким образом избавимся в них от x_2 (от x_1 мы в них избавились раньше). Затем из 3-го уравнения определим x_3 и т.д. пока в последнем уравнении останется одно единственное неизвестное x_n. Это так называемый прямой ход алгоритма Гаусса (последовательное исключение неизвестных из уравнений системы – откуда и название метода).

А дальше – очевидно. Из последнего (n-го) уравнения определяем x_n (что может быть проще?). Из (n–1)-го теперь можно определить x_{n-1} (x_n то уже известно – только что определили), затем из (n–2)-го определяем x_{n-2} и пошло-поехало – вплоть до x_1. Это обратный ход алгоритма Гаусса.

Алгоритм прост и понятен. Правда, иногда могут возникать проблемы. Ну, скажем, надо определить x_i из i-го уравнения, а его, как назло, там нет. Ну нет – и все тут, $a_{i,i}$=0. Так что – приехали? А может все-таки это еще не конец света? Ну, ладно, в i-м уравнении x_i нет, но зато может среди остальных он где-то завалялся? А ну, давайте поищем. Если найдем, скажем, в $z$max-ой строке, то поменяем местами i-ю и $z$max-ю строки (решение системы уравнений от перестановки в ней уравнений не изменится, верно?). Это выход из положения. Ну, а если не найдется? Ну нет x_i-го во всех оставшихся уравнениях! На нет – и суда нет. Раз его там нет, то определить x_i неоткуда. Это, действительно, тупик! Раньше мы уже обсуждали возможность отсутствия решения. Бывают, значит, положения, из которых нет выхода … – придется так и записать: решение не найдено (писать решения нет – как-то слишком категорично звучит, надо быть скромнее, да?).

И еще один существенный аспект. Вот мы говорим нет $a_{i,i}$ отличного от нуля. А мы не забыли, что $a_{i,i}$ не само по себе возникло, а является, возможно, результатом каких-угодно (и скольких угодно) операций над вещественными числами (не говоря уже о том, что и исходные значения коэффициентов системы , вероятнее всего, были определены не бог весть как точно). Итог – это $a_{i,i}$ известно нам с погрешностью. Если погрешность близка к модулю $a_{i,i}$, то стоит ли доверять этому $a_{i,i}$? Вот, станем мы определять x_i из (i-го) уравнения, ведь для этого все остальные (кроме $a_{i,i}$) коэффициенты i-го уравнения придется делить на $a_{i,i}$. А что такое делить на малое по модулю (в том числе, вероятно, равное нулю) число – читатель уже понял? Получается, что, если $|a_{i,i}|<\varepsilon$ (ε – малое число), то это все равно, что его нет (практически нет) – искать надо получше (побольше по модулю). Но раз уж все равно искать, то, может, стоит искать не просто уравнение, где такой коэффициент при x_i больше (по модулю) чем ε , а такое уравнение, где такой коэффициент был бы максимальным по модулю. Это называется – выбор главного элемента, и метод в этом случае уже называется: методом Гаусса с выбором главного элемента.

Договоримся «складировать» найденные значения решения (x_1, x_2, …, x_n) в массиве *X:Coef*, а их общее количество отражать в ячейке *X[-1]*. Если решение не состоялось (бывает) – тогда в *X[-1]* запишем 0.

```
procedure SystUr(n:integer; A:Matr; var x:Coef);
  const Eps=1e-9;
  var Zmax,z,s,i,j:integer;
      r:real;
      b:boolean;
  begin
   b:=true;
   if n=1
```

```
then
  if abs(A[1,1])<Eps
   then b:=false else X[1]:=A[1,2]/A[1,1]
else
 begin
   i:=1;
   while (i<=n-1) and b do
  begin
     Zmax:=i;
     for z:=i+1 to n do
      if abs(A[z,i])>abs(A[Zmax,i])
         then Zmax:=z;
      if abs(A[Zmax,i])<Eps
        then b:=false
       else
      begin
        if Zmax>i then
         for j:=i to n+1 do
          begin
            r:=A[i,j];
            A[i,j]:=A[Zmax,j];
            A[Zmax,j]:=r
          end;
        for z:=i+1 to n do
          if abs(A[z,i])>Eps then
          begin
             r:=A[z,i]/A[i,i];
             for s:=i+1 to n+1 do
            A[z,s]:=A[z,s]-r*A[i,s]
          end
       end;
       inc(i)
  end;
  if b then
   if abs(A[n,n])<Eps then b:=false;
  if b then
   begin
    X[n]:=A[n,n+1]/A[n,n];
    for i:=n-1 downto 1 do
     begin
      r:=0;
      for s:=i+1 to n do
       r:=r+A[i,s]*X[s];
      X[i]:=(A[i,n+1]-r)/A[i,i]
     end
```

```
      end
    end;
  if b then X[-1]:=n else X[-1]:=0
 end;
```

Здесь *A:Matr* – расширенная матрица коэффициентов системы уравнений вида , *n* – порядок системы (*n* уравнений с *n* неизвестными), *X:Coef* – массив решений, где x_i=*X[i]*, 1≤*i*≤*n*. В *X[-1]* – это *n*, если решение системы найдено, или *X[-1]*=0, когда решения нет.

Константа *Eps*=1e-9 представляет ε. Если «наиболее достойный предентент» на занятие ячейки *A[i,i]*, (1≤*i*≤*n*). по модулю не превышает *Eps*, то процесс решения прекращается – *X[-1]*:=0.

Переменная *b:boolean* – флажок. Пока *b:=true* (флаг гордо реет) – надежда на победу не потеряна – сражение продолжается. Если *b:=false* – игра проиграна, выходим из игры – *X[-1]*:=0.

Поиск решения начинается с установки флага – *b:=true*.

Если *n*=1, система сводится к одному уравнению вида

$$a_{1,1}x_1=a_{1,2},$$

решение которого очевидно $x_1=a_{1,2}/a_{1,1}$, которое воплощается в жизнь при $|a_{1,1}|>\varepsilon$, иначе *b:=false*.

Если же условие *n*=1 не выполняется, то задействуется алгоритм Гаусса.

i:integer – это счетчик обслуживаемых диагональных элементов. «Обслуживание» состоит в обнулении элементов массива *A*, расположенных под данным диагональным. Этим занимается цикл

```
while (i<=n-1) and b do …
```

Перед входом в цикл *i* получает свое начальное значение (*i*:=1). В теле цикла *i* последовательно наращивается с шагом 1 (см. оператор *inc(i)*), последний раз тело цикла выполняется при *i*=*n*-1, т.к., при *i*=*n*, нечего обнулять под *n*-м диагональным.

Тело цикла *while* начинается с поиска *Zmax* – номера строки, в которой элемент *A[Zmax,i]* будет наибольшим по модулю среди *A[z,i]*, *i*≤*z*≤*n*. Найденный *A[Zmax,i]* по модулю сравнивается с *Eps*. Если он оказался слабаком – *b:=false*. И только тогда, кода он признан достойным занять позицию *A[i,i]*, строки *Zmax*-ая и i-ая меняются местами.

Но, понятно, перестановку имеет смысл затевать, если *Zmax* не равняется *i* (начинали-то поиск мы с *Zmax:=i*).

Обнулением элементов *i*-го столбца занимается цикл

```
for z:=i+1 to n do …
```

Опять же стоит ли обнулять элемент *A[z,i]*, если он и так «на ладан дышит»

```
(if abs(A[z,i])<Eps).
```

Когда «обслуживание» i-го диагонального элемента закончено, можно переходить к следующему (*inc(i)*).

Таким образом, цикл *while*, по сути, реализует прямой ход алгоритма Гаусса.

Правда, после выхода из *while* необходимо произвести последнюю проверку – сравнить с *Eps* элемент *A[n,n]* – этот элемент в цикле не рассматривался как диагональный.

Обратный ход имеет смысл только в случае успешного завершения прямого (*b=true*).

Начинается обратный ход с определения x_n, а затем в цикле

for i:=n-1 downto 1 do …

вычисляется r – сумма слагаемых с уже найденными значениями неизвестных (с номерами от *i*+1 до *n*), после чего в *X[i]* заносится значение очередной неизвестной.

Завершается процедура *SystUr* заполнением ячейки *X[-1]*, как уже ранее обсуждалось.

Проверка работоспособности (корректности) процедуры *SystUr* осуществляется программой *OSystUr*, листинг которой приводится ниже.

```
program OSystUr;
  uses Crt, Graph, Serv;
  const Nmax=10;
  type Matr=array[1..Nmax, 1..Nmax+1] of real;
  var Xc, X:Coef;
   A:Matr;
   z,s,n:integer;
   Sum:real;
   T30:string[30];
   {$i SystUr.pas}
    begin
      repeat
        PutA; Ou('Esc-exit, 1-System');
        J:=ReadKey;
       Case J of
        '1':begin
              ClearDevice;
              Oui('n-order of System',n);
              PutA; Ou('Enter Contol Solutions');
              Delay(3000); Xc[-1]:=n;
              for s:=1 to n do
                begin
                   Str(s,T10);
                  Our('Xc['+T10+']',Xc[s])
                end;
               PutA; Ou('Enter Coefficients of the System');
              Delay(3000);
              for z:=1 to n do
                begin
                  Sum:=0; Str(z,T10);
                  for s:=1 to n do
                   begin
                     Str(s,T30);
                     Our('A['+T10+','+T30+']',A[z,s]);
```

```
            Sum:=Sum+A[z,s]*Xc[s]
           end;
           A[z,n+1]:=Sum
         end;
         SystUr(n,A,X);
         for s:=1 to n do
         begin
            Str(s,T10);Str(Xc[s],T30);
            OutTextXY(50,(s+3)*10,'Xc['+T10+']='+T30)
          end;
         for s:=1 to round(X[-1]) do
        begin
          Str(s,T10);Str(Xc[s],T30);
          OutTextXY(320,(s+3)*10,'X['+T10+']='+T30)
         end;
       if X[-1]=0 then
         OutTextXY(320,30,'Solution is not found')
      end;
    end;
   Until J=#27;
  CloseGraph
end.
```

Программа *OSystUr*, как уже отмечалось, предназначена для проверки работы процедуры, следовательно, подразумевается, что добросовестный и квалифицированный студент предлагает программе систему с известными значениями решения и сопоставляет эти решения с теми, что, будем надеяться, найдет программа.

После того как студент ответил на вопрос, какого порядка n он хотел бы ввести систему, предлагаем ему придумать ответ, т.е. ввести массив *Xc* – значения X_1, X_2, X_3, …, X_n. И только после этого у него выспрашиваем по одному коэффициенты левой части каждого из уравнений системы. По мере ввода этих коэффициентов для каждой строки (уравнения) накапливается сумма произведений коэффициентов на значения стоящих после них неизвестных, контрольные значения которых берутся из массива *Xc*. А вот правую часть уравнения вообще не спрашиваем, а просто молча вписываем туда накопленную сумму. Дальше – *SystUr* – распечатка контрольного решения слева, и справа – найденное решение для сопоставления. Если решения нет – сообщение «Solution is not found».

Інтерполяція поліномами. Нехай маємо множину точок x_s, y_s, $0 \le s \le m$ на площині *xoy*.

Поліном, графік якого проходить через усі точки даної множини, називається інтерполяційним. Нехай, це буде поліном

$$A(x) = \alpha_n x^n + \alpha_{n-1} x^{n-1} + ... + \alpha_1 x + \alpha_0 .$$

Для того, щоб графік полінома *A(x)* гарантовано пройшов через усі точки вказаної множини, його степінь n має дорівнювати m. Умови проходження графіка через кожну точку множини

$$\begin{cases} \alpha_0 + \alpha_1 x_0 + \alpha_2 x_0^2 + ... + \alpha_n x_0^n = y_0 , \\ \alpha_0 + \alpha_1 x_1 + \alpha_2 x_1^2 + ... + \alpha_n x_1^n = y_1 , \\ \alpha_0 + \alpha_1 x_2 + \alpha_2 x_2^2 + ... + \alpha_n x_2^n = y_2 , \\ ..... \\ \alpha_0 + \alpha_1 x_n + \alpha_2 x_n^2 + ... + \alpha_n x_n^n = y_n . \end{cases}$$

Ці умови утворюють систему з *(n+1)* рівнянь з *(n+1)* невідомими коефіцієнтами полінома.

Розширена матриця *Rm* цієї системи матиме структуру

	1	2	3	4	…	*n*+1	*n*+2
1	1	x_0	x_0^2	x_0^3	…	x_0^n	y_0
2	1	x_1	x_1^2	x_1^3	…	x_1^n	y_1
3	1	x_2	x_2^2	x_2^3	…	x_2^n	y_2
…	…	…	…	…	…	…	…
n+1	1	x_n	x_n^2	x_n^3	…	x_n^n	y_n
	α_0	α_1	α_2	α_3		α_n	

Під стовпчиками розширеної матриці *Rm* вказані невідомі, коефіцієнтами, при яких є елементи даного стовпчика.

Формування матриці *Rm* та визначення коефіцієнтів $\alpha_0, \alpha_1, ..., \alpha_n$ (масиву *A*) – процедура *IntPol*.

```
procedure IntPol (x,y:CoefR; var A:Coef);
 var z,s,n:integer;
   begin
     n:=round(X[-1]);
     for z:=1 to n+1 do
```

```
    begin
      D[z,1]:=1; D[z,n+2]:=Y[z-1];
      for s:=2 to n+1 do
        D[z,s]:=D[z,s-1]*X[z-1]
    end;
  SystUr(n+1,D,A);
  if A[-1]>0 then
    begin
      A[-1]:=n;
      for s:=0 to n do A[s]:=A[s+1]
    end
end;
```

В демонстраційно-відлагоджувальній програмі *ApprPol*, що наводиться у наступному підрозділі, множину точок для простоти будемо формувати як значення контрольного полінома на рівномірній сітці значень X (між x_n та x_k з кроком $h_x = (x_k - x_n)/m$). Знайдений за допомогою процедури *IntPol* інтерполяційний поліном може співставлятись з контрольним порівнянням відповідних коефіцієнтів або шляхом нанесення графіка інтерполяційного полінома на точки згаданої множини (на графіку зображуються кружечками).

Лекція 5. Метод найменших квадратів

Нехай маємо множину точок x_s, y_s, $0 \le s \le m$ на площині *xoy*.

Треба сформувати поліном *n*-го степеня, графік якого пройшов би по можливості якомога ближче до точок вказаної множини (апроксимувати множину-поліномом).

За показчик якості апроксимації приймаємо:

$$E = \sum_{S=0}^{m} \left(\alpha_0 + \alpha_1 x_S + \alpha_2 x_S^2 + ... + \alpha_n x_S^n - y_S\right)^2 .$$

Коефіцієнти апроксимуючого полінома будемо вибирати з умови, щоб зробити E мінімальним, тобто вимагаємо щоб:

$$\begin{cases} \dfrac{\partial E}{\partial \alpha_0} = 0, \\ \dfrac{\partial E}{\partial \alpha_1} = 0, \\ \dfrac{\partial E}{\partial \alpha_2} = 0, \\ \ldots \\ \dfrac{\partial E}{\partial \alpha_n} = 0. \end{cases}$$

Якщо розшифрувати значення похідних, то одержимо систему рівнянь:

$$\begin{cases} \sum\limits_{S=0}^{m} \left(\alpha_0 + \alpha_1 x_S + \alpha_2 x_S^2 + \ldots + \alpha_n x_S^n - y_S\right) x_S^0 = 0, \\ \sum\limits_{S=0}^{m} \left(\alpha_0 + \alpha_1 x_S + \alpha_2 x_S^2 + \ldots + \alpha_n x_S^n - y_S\right) x_S^1 = 0, \\ \sum\limits_{S=0}^{m} \left(\alpha_0 + \alpha_1 x_S + \alpha_2 x_S^2 + \ldots + \alpha_n x_S^n - y_S\right) x_S^2 = 0, \\ \qquad\qquad \ldots \\ \sum\limits_{S=0}^{m} \left(\alpha_0 + \alpha_1 x_S + \alpha_2 x_S^2 + \ldots + \alpha_n x_S^n - y_S\right) x_S^n = 0. \end{cases}$$

Якщо розглядати $\alpha_0, \alpha_1, \alpha_2, \ldots, \alpha_n$ як невідомі, то вищезаписана система матиме розширену матрицю такої структури:

	1	2	3	…	n+1	n+2
z=1	$\sum\limits_{S=0}^{m} x_S^0$	$\sum\limits_{S=0}^{m} x_S^1$	$\sum\limits_{S=0}^{m} x_S^2$	…	$\sum\limits_{S=0}^{m} x_S^n$	$\sum\limits_{S=0}^{m} y_S x_S^0$
z=2	$\sum\limits_{S=0}^{m} x_S^1$	$\sum\limits_{S=0}^{m} x_S^2$	$\sum\limits_{S=0}^{m} x_S^3$	…	$\sum\limits_{S=0}^{m} x_S^{n+1}$	$\sum\limits_{S=0}^{m} y_S x_S^1$
z=3	$\sum\limits_{S=0}^{m} x_S^2$	$\sum\limits_{S=0}^{m} x_S^3$	$\sum\limits_{S=0}^{m} x_S^4$	…	$\sum\limits_{S=0}^{m} x_S^{n+2}$	$\sum\limits_{S=0}^{m} y_S x_S^2$
…	...	...	...	…	...	...

n+1	$\sum_{S=0}^{m} x_S^n$	$\sum_{S=0}^{m} x_S^{n+1}$	$\sum_{S=0}^{m} x_S^{n+2}$	…	$\sum_{S=0}^{m} x_S^{2n}$	$\sum_{S=0}^{m} y_S x_S^n$
	α_0	α_1	α_2	…	α_n	

Під стовпчиками розширеної матриці показані невідомі, коефіцієнтами при яких є елементи відповідного стовпчика.

Формування розширеної матриці та визначення апроксимуючого полінома реалізується в підпрограмі *NaimKv*, а розглянутий алгоритм носить назву методу найменших квадратів (МНК).

```
procedure NaimKv(x,y:CoefR;n:integer; var e:real; var
                 A:Coef);
var m,z,s:integer;
     Sum:real;
     B,C:CoefR;
  begin
    m:=round(X[-1]);
    for s:=0 to m do B[s]:=1;
    for z:=0 to 2*n do
      begin
        C[z]:=0; Sum:=0;
        for s:=0 to m do
          begin
            C[z]:=C[z]+B[s];
            if Z<=n then Sum:=Sum+B[s]*Y[s];
            B[s]:=B[s]*X[s]
          end;
        if z<=n then D[z+1,n+2]:=Sum
      end;
    for z:=1 to n+1 do
      for s:=1 to n+1 do
```

```
        D[z,s]:=C[z+s-2];
    SystUr(n+1,D,A); A[-1]:=n; e:=0;
    for s:=0 to n do A[s]:=A[s+1];
    for s:=0 to m do e:=e+sqr(HorReal(A,X[s])-Y[s])
end;
```

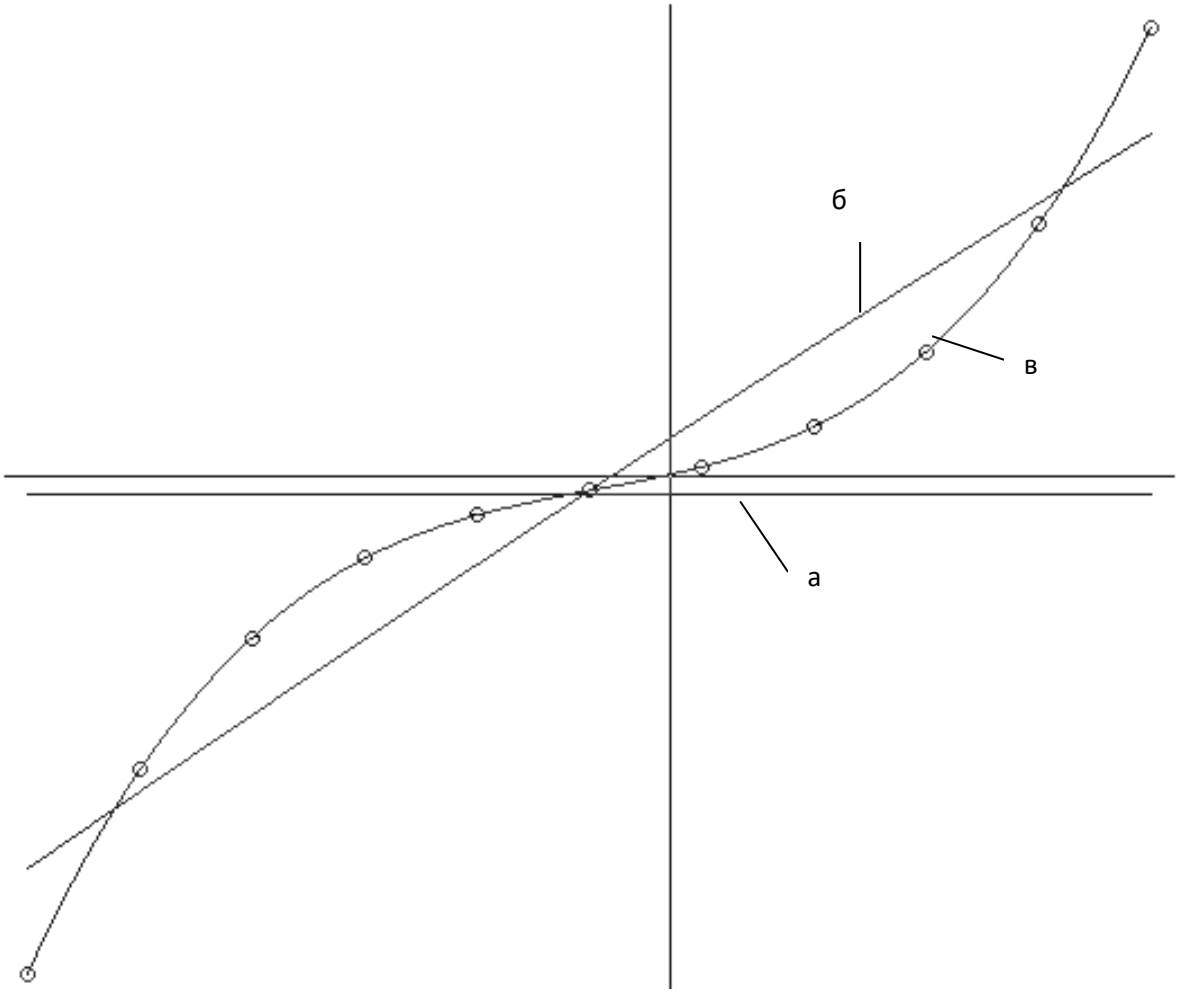

Рис. 5. Згладжування методом найменших квадратів для: а)*n*=0;б) *n*=1;в)*n*=2

Розділ 2. Сплайни.

Лекція 6. Інтерполяційні кубічні сплайни

Задана множина значень аргументу x (вузлів апроксимації)

$x_0<x_1<x_2<..<x_{m-1}<x_m$

і кожному вузлу поставлено у відповідність значення функції

$y_0, y_1, y_2, .., y_{m-1}, y_m$.

Цей набір даних в площині *XOY* можна представити рядом точок з координатами (x_S, y_S), $0 \le s \le m$.

Функціональну залежність y від x будемо шукати у вигляді кубічного полінома типу

$$\psi_s(x) = a_{0,s} + a_{1,s}(x - x_{s-1}) + a_{2,s}(x - x_{s-1})^2 + a_{3,s}(x - x_{s-1})^3,\ 1 \le s \le m.$$

Перша та друга похідні від

$$\psi_s'(x) = a_{1,s} + 2a_{2,s}(x - x_{s-1}) + 3a_{3,s}(x - x_{s-1})^2, 1 \le s \le m,$$

$$\psi_s''(x) = 2a_{2,s} + 6a_{3,s}(x - x_{s-1}),\ 1 \le s \le m,$$

Тут індекс s означає, що поліном відноситься до s-го інтервалу між вузлами апроксимації (s-ий інтервал починається в точці x_{S-1}, закінчується – в x_S).

Отже, поліномів буде m – по одному на кожен з m інтервалів.

Система поліномів описує сплайн (гнучку лінійку), яка проходить через кожну із згаданих точок на площині *XOY*.

Коефіцієнти $a_{0,S}$, $a_{1,S}$, $a_{2,S}$, $a_{3,S}$ полінома підбираються таким чином, щоб забезпечити на інтервалі x_0 .. x_m неперервність не тільки апроксимуючої функції , але і її першої та другої похідних.

Неперервність забезпечується тим, що сплайн має проходити через кожну із згаданих точок, а кожна точка (крім крайніх) є кінцем одного і початком наступного інтервалів. Отже, умова неперервності на лівому кінці s-го інтервалу (в точці x_{s-1}) вимагає, щоб

$\psi_s(x_{s-1}) = y_{s-1}$,

тобто згідно з

$a_{0,s} = y_{s-1}, 1 \le s \le m.$

На правому кінці s-го інтервалу сплайн має пройти через точку з координатами (x_S, y_S), отже, згідно з

$a_{0,s} + a_{1,s}h_s + a_{2,s}h_s^2 + a_{3,s}h_s^3 = y_s, 1 \le s \le m,$

де h_S-ширина s-го інтервалу, $h_S = x_S - x_{S-1}$.

Умова неперервності перших похідних на стиках інтервалів (в точці x_S, яка є кінцем s-го та початком (s+1)-го інтервалів) згідно з

$a_{1,s} + 2a_{2,s}h_s + 3a_{3,s}h_s^2 = a_{1,s+1}, 1 \le s \le m-1,$

Для других похідних аналогічно згідно

$a_{2,s} + 3a_{3,s}h_s = a_{2,s+1}, 1 \le s \le m-1.$

Рівняння визначають коефіцієнти, першим індексом яких є 0, отже $a_{0,S}$ ($1 \le s \le m$) можна вважати уже визначеними.

Рівняння ж - можна розглядати як систему з ($3m$-2) рівнянь відносно $3m$ невідомих коефіцієнтів $a_{1,S}, a_{2,S}, a_{3,S}$, $1 \le s \le m$.

Якщо згадану систему доповнити двома рівняннями граничних умов (мова про них піде нижче), то матимемо $3m$ рівнянь відносно $3m$ невідомих, яку можна спробувати розв'язати.

Визначимо спочатку $a_{3,S}$ з

$$a_{3,s} = \frac{\left(a_{2,s+1} - a_{2,s}\right)}{3h_s}$$

та підставимо в та , позбавившись в від $a_{0,S}$ у відповідності з . Після спрощення одержимо

$3a_{1,s}h_s + 2a_{2,s}h_s^2 + a_{2,s+1}h_s^2 = 3\left(y_s - y_{s-1}\right),$

$a_{1,s} + a_{2,s}h_s + a_{2,s+1}h_s = a_{1,s+1}$

Тоді з визначаємо $a_{1,S}$

$$\alpha_{1,s} = \frac{y_s - y_{s-1}}{h_s} - \frac{2}{3}\alpha_{2,s}h_s - \frac{1}{3}\alpha_{2,s+1}h_s.$$

Підставляємо в замість $a_{1,S}$ та $a_{1,S+1}$ та спрощуємо

$h_s a_{2,s} + e_s a_{2,s+1} + h_{s+1}a_{2,s+2} = r_s$, $1 \le s \le m-1,$

де

$$e_s = 2(h_s + h_{s+1}),$$

$$r_s = 3\left(\frac{y_{s+1} - y_s}{h_{s+1}} - \frac{y_s - y_{s-1}}{h_s}\right).$$

Система є тридіагональною, і ми будемо її розв'язувати методом прогонки.В процесі прямого ходу будемо приводити рівняння цієї системи до вигляду

$a_{2,s+1} = A_s a_{2,s+2} + B_s$, $1 \le s \le m-1$.

Підставляємо у замість $a_{2,S}$. Результат представимо в формі

$$\alpha_{2,s+1} = \frac{-h_{s+1}}{e_s + A_{s-1}h_s}\alpha_{2,s+2} + \frac{r_s - B_{s-1}h_s}{e_s + A_{s-1}h_s}.$$

Порівнюючи з , одержуємо рекурентні формули для A_s та B_s

$$A_s = \frac{-h_{s+1}}{e_s + A_{s-1}h_s},$$

$$B_s = \frac{r_s - B_{s-1}h_s}{e_s + A_{s-1}h_s}.$$

Щоб формулами , можна було скористатись, треба спочатку визначити A_0 та B_0. Використаємо для цього граничну умову на лівому кінці слайна. Розглянемо два найпростіших варіанти умов на кінцях слайна: 1-го типу, коли задається значення першої похідної та 2-го типу – зі значенням другої похідної на відповідному кінці.

Так, наприклад, якщо на лівому кінці сплайну перша похідна дорівнює y'_0, то, згідно з для точки X_0 при s=1 матимемо

$a_{1,1} = y'_0$.

Підставляємо у при s=1

$$\frac{y_1 - y_0}{h_1} - \frac{2}{3}\alpha_{2,1}h_1 - \frac{1}{3}\alpha_{2,2}h_1 = y'_0,$$

Звідки

$$\alpha_{2,1} = -\frac{1}{2}\alpha_{2,2} + \frac{\frac{3}{2}\left(\frac{y_1 - y_0}{h_1} - y'_0\right)}{h_1}.$$

Порівнюємо з при s=0, звідки

$$\begin{cases} A_0 = -\dfrac{1}{2}, \\ B_0 = \dfrac{\dfrac{3}{2}\left(\dfrac{y_1 - y_0}{h_1} - y'_0 \right)}{h_1}. \end{cases}$$

Якщо ж на лівому кінці сплайну задано значення другої похідної, тобто y''_0, то згідно з для точки X_0 при s=1 матимемо

$$2a_{2,1} = y''_0.$$

Порівнюючи з при s=0, одержимо

$$\begin{cases} A_0 = 0, \\ B_0 = \dfrac{y''_0}{2}. \end{cases}$$

Маючи A_0, B_0, за формулами , послідовно визначаються A_s, B_s, $1 \le s \le m-1$.

При s=m-1 формула дає

$$a_{2,m} = A_{m-1}a_{2,m+1} + B_{m-1}.$$

Щоб почати зворотній хід, скористаємось граничною умовою на правому кінці сплайну. Нехай на правому кінці маємо граничну умову 1-го роду

$$\psi'_m(x_m) = y'_m.$$

Якщо у підставити при s=m, то одержимо

$$a_{1,m} + 2a_{2,m}h_m + 3a_{3,m}h_m^2 = y'_m.$$

Коли ми тепер замінимо в $a_{1,m}$ за формулою при s=m, а $a_{3,m}$ за формулою при s=m, то після спрощення матимемо

$$\alpha_{2,m}h_m + 2\alpha_{2,m+1}h_m = 3\left(y'_m - \frac{y_m - y_{m-1}}{h_m} \right).$$

Розв’язуємо систему рівнянь, яку утворюють та

$$\alpha_{2,m+1} = \frac{3\left(y'_m - \dfrac{y_m - y_{m-1}}{h_m} \right) - B_{m-1}h_m}{(A_{m-1} + 2)h_m}.$$

Коли ж на правому кінці сплайна маємо граничну умову 2-го роду

$$\psi''_m(x_m) = y''_m,$$

то, враховуючи , матимемо

$$2a_{2,m} + 6a_{3,m}h_m = y''_m,$$

а коли врахувати при $s=m$, то одержимо

$$\alpha_{2,m+1} = \frac{y''_m}{2}.$$

Після визначення $a_{2,m+1}$ зворотний хід реалізуємо за формулою , змінюючи s від $s=m$-1 до 1 включно.

Цим завершується визначення масиву коефіцієнтів $a_{2,S}$, $1 \le s \le m$.

За формулами та для усіх потрібних s (від 1 до m) визначаються $a_{3,S}$ та $a_{1,S}$ (коефіцієнти $a_{0,S}$ були визначені формулою).

Описаний алгоритм реалізується підпрограмою *SplineN*.

```
procedure SplineN
(X,Y:CoefR;Ngl,Ngr:integer;D0,Dm:real;
A0,A1,A2,A3:CoefR);
var m,s:integer;
    H,H1,E,R,V,Z:real;
    A,B:CoefR;
begin
  m:=round(X[-1]);
  case Ngl of
    1: begin
         H:=X[1]-X[0]; A[0]:=-0.5;
         B[0]:=1.5*((Y[1]-Y[0])/H-D0)/H
       end;
    2: begin
         A[0]:=0; B[0]:=D0/2
       end
  end;
```

```
  for s:=1 to m-1 do
    begin
      H:=X[s]-X[s-1]; H1:=X[s+1]-X[s];
      E:=2*(H+H1);
      R:=3*((Y[s+1]-Y[s])H1-(Y[s]-Ys[S-1])/H);
      Z:=E+A[s-1]*H; A[s]:=-H1/Z;
      B[s]:=(R-B[s-1]*H)/Z
    end;
  case Ngr of
    1: begin
         H:=X[m]-X[m-1];
         A2[m+1]:=(3*(Dm-(Y[m]-Y[m-1])/H)-B[m-1]*H)/
         ((A[m-1]+2)*H)
       end;
    2: A2[m+1]:=Dm/2
  end;
  for s:=m-1 downto 0 do
    A2[s+1]:=A[s]*A2[s+2]+B[s];
  for s:=1 to m do
    begin
      H:=X[s]-X[s-1];
      A1[s]:=(Y[s]-Y[s-1])/H-2/3*A2[s]*H-1/3*A2[s+1]*H;
      A3[s]:=(A2[s+1]-A2[s])/(3*H); A0[s]:=Y[s-1]
    end;  A0[-1]:=m
end;
```

Після того, як сплайн сформовано (тобто сформовано масиви *A*0, *A*1, *A*2, *A*3), інтерполяція виконується підпрограмою-функцією *SplintN*, яка фактично реалізує формулу , визначаючи спочатку значення *s*-номер інтервалу, в який попадає замовлене значення *x*.

```
function SplintN(A0,A1,A2,A3,Mx:CoefR; X:real):real;
```

```
var m,s:integer;
  begin
    s:=1;
    while Mx[s-1] < x then inc(s); x:=x-Mx[s-1];
    SplintN:=((A3[s]*x+A2[s])*x+A1[s])*x+A0[s]
  end;
```

При практичному використанні часто виникає ситуація, коли вузли апроксимації утворюють арифметичну прогресію зі сталим кроком h_S=const. Тоді можна обійтись без масиву *Mx*, а щоб іще спростити задачу, можна за вузли апроксимації прийняти номери цих вузлів, тобто нормувати *x* таким чином, щоб він приймав значення в діапазоні 0..*m*.

Якщо крім того розглядати масиви *A*0,*A*1,*A*2,*A*3 як глобальні, а граничні умови на кінцях сплайну за умовчанням вважати умовами 2-го роду при нульвих значеннях других похідних, то спрощений варіант підпрограми *SplineN* можна представити як підпрограму *Spline*.

```
procedure Spline(Y:CoefR);
var m,s:integer;
    Z,R:real;
begin
  m:=round(Y[-1]); A0[-1]:=m;A0[0]:=0;
  A1[0]:=0;
  for s:=1 to m-1 do
    begin
      R:=3*(Y[s+1]-2*Y[s]+Y[s-1]);
      Z:=4+A0[s-1];A0[s]:=-1/Z;
      A1[s]:=(R-A1[s-1])/Z
    end;
  A2[m+1]:=0;
  for s:=m-1 downto 0 do
    A2[s+1]:=A0[s]*A2[s+2]+A1[s];
```

```
  for s:=1 to m do
    begin
      A1[s]:=Y[s]-Y[s-1]-(2*A2[s]+A2[s+1])/3;
      A3[s]:=(A2[s+1]-A2[s])/3;A0[s]:=Y[s-1]
    end;
end;
```

Функція *SplintN* в подібній ситуації набуває вигляду (для нормованого *x*)

```
function Splint(x:real):real;
var m,s:integer;
begin
  s:=trunc(x)+1; x:=frac(x);
  Splint:=((A3[s]*x+A2[s])*x+A1[s])*x+A0[s]
end;
```

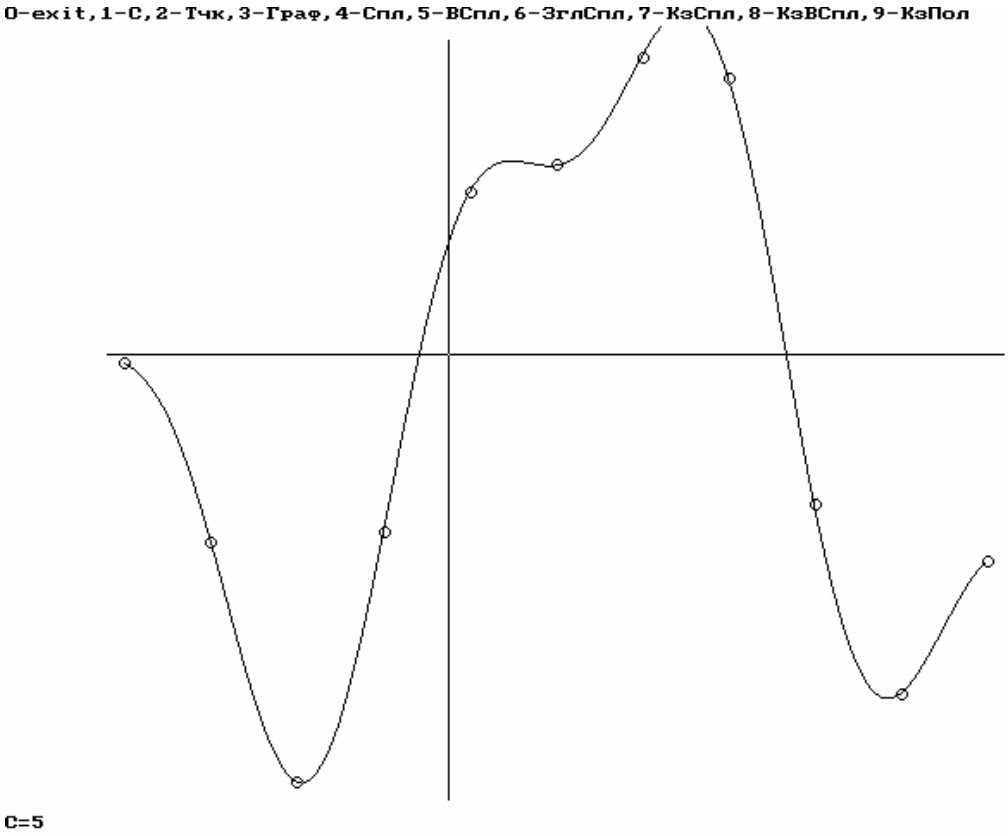

Рис. 6. Апроксимація за допомогою інтерполяційного кубічного сплайна

Лекція 7. Згладжуючі кубічні сплайни

Як і в попередніх параграфах будемо вважати, що задана множина вузлів апроксимації

$x_s=sh$, $s=0,1,2,\ldots,m$ (крок h - сталий)

та відповідних їм значень функції

y_s, $s=0,1,2,\ldots,m$.

Ставиться задача сформувати апроксимуючу функцію у вигляді кубічного сплайну типу

$$\psi_s(x)=a_{0,s}+a_{1,s}(x-x_{s-1})+a_{2,s}(x-x_{s-1})^2+a_{3,s}(x-x_{s-1})^3.$$

Від цього сплайну будемо вимагати, щоб його графік пройшов по можливості близько до точок (x_s,y_s), $1\le s\le m$ і щоб перша та друга похідна від сплайну були б неперервні в діапазоні $[x_0 \,..\, x_m]$.

Отже, сплайн має згладжувати досліджувану залежність. Порівняно з інтерполяційним кубічним сплайном, розглянутим вище, даний сплайн повинен формувати більш гладку залежність. Для визначення коефіцієнтів сплайна розкладемо досліджувану функцію $y(x)$ в ряд Тейлора в околі точки x_{s-1}, обмежившись першими чотирма доданками

$$y(x)=y\Big|_{x=x_{s-1}}+\frac{dy}{dx}\Big|_{x=x_{s-1}}(x-x_{s-1})+\frac{1}{2!}\frac{d^2y}{dx^2}\Big|_{x=x_{s-1}}(x-x_{s-1})^2+\frac{1}{3!}\frac{d^3y}{dx^3}\Big|_{x=x_{s-1}}(x-x_{s-1})^3$$

Порівнюючи з , бачимо, що коефіцієнти $a_{0,s},a_{1,s},a_{2,s},a_{3,s}$ можуть бути виражені таким чином

$$\begin{cases} a_{0,s}=y\Big|_{x=x_{s-1}}, \\ a_{1,s}=\dfrac{dy}{dx}\Big|_{x=x_{s-1}}, \\ a_{2,s}=\dfrac{1}{2}\dfrac{d^2y}{dx^2}\Big|_{x=x_{s-1}}, \\ a_{3,s}=\dfrac{1}{3!}\dfrac{d^3y}{dx^3}\Big|_{x=x_{s-1}}. \end{cases}$$

Формула для a_0 в вимагає, щоб сплайн проходив через точку

(x_{s-1}, y_{s-1}), тобто був би інтерполяційним. Оскільки мова йде про згладжуючий сплайн, то цю умову треба пом'якшити.

Приймемо, що

$$a_{0,s} = \frac{y_{s-2} + 4y_{s-1} + y_s}{6} \quad ,$$

тоді як інші коефіцієнти визначимо через різницеві співвідношення, що наближено представляють похідні в формулах

$$a_{1,s} = \frac{y_s - y_{s-2}}{2h},$$

$$a_{2,s} = \frac{y_{s-2} - 2y_{s-1} + y_s}{2h^2},$$

$$a_{3,s} = \frac{-y_{s-2} + 3y_{s-1} - 3y_s + y_{s+1}}{6h^3},$$

де h - крок за x (вважаємо його сталим, що було відзначено на початку параграфа, $x_s = sh$)

$$h = x_s - x_{s-1}.$$

Доцільність вибору саме таких формул для коефіцієнтів сплайну буде підтверджена нижче.

Перевіримо спочатку неперервність функції на межах інтервалів (s-тий інтервал знаходиться між точками x_{s-1} та x_s).

На кінці s-того інтервалу (в точці x_s) згідно з формулами та - матимемо

$$\psi_s(x_s) = \frac{y_{s-2} + 4y_{s-1} + y_s}{6} + \frac{y_s - y_{s-2}}{2h} h + \frac{y_{s-2} - 2y_{s-1} + y_s}{2h^2} h^2 +$$
$$+ \frac{-y_{s-2} + 3y_{s-1} - 3y_s + y_{s+1}}{6h^3} h^3,$$

що після спрощення дає

$$\psi_s(x_s) = \frac{y_{s-1} + 4y_s + y_{s+1}}{6}.$$

На початку ж (s+1)-го інтервалу (у тій же точці x_s) матимемо відповідно

$$\psi_{s+1}(x_s) = \frac{y_{s-1} + 4y_s + y_{s+1}}{6}.$$

Тут $\psi_{s+1}(x_s) = a_{0,s+1}$,оскільки дужки в формулі стають рівними нулю при заміні s на s+1 та при x=x_s.

Порівняння та підтверджує неперервність функції на стику s-того та (s+1)-го інтервалів (в точці x_s).

Продиференціюємо відносно x

$$\psi_s'(x) = a_{1,s} + 2a_{2,s}(x - x_{s-1}) + 3a_{3,s}(x - x_{s-1})^2.$$

Обчислимо значення $\psi_s'(x)$ на кінці s-го інтервалу (в точці x_s)

$$\psi_s'(x_s) = \frac{y_s - y_{s-2}}{2h} + 2\frac{y_{s-2} - 2y_{s-1} + y_s}{2h^2}h + 3\frac{-y_{s-2} + 3y_{s-1} - 3y_s + y_{s+1}}{6h^3}h^2,$$

що після спрощення дає

$$\psi_s'(x_s) = \frac{y_{s+1} - y_{s-1}}{2h}.$$

На початку (s+1)-го інтервалу

$$\psi_{s+1}'(x_s) = a_{1,s+1},$$

тобто, згідно з при заміні s на s+1

$$\psi_{s+1}'(x_s) = \frac{y_{s+1} - y_{s-1}}{2h}.$$

Порівняння з підтверджує неперервність першої похідної від сплайнової кривої на стику s-го та (s+1)-го інтервалів.

Друга похідна

$$\psi_s''(x) = 2a_{2,s} + 6a_{3,s}(x - x_{s-1}).$$

На кінці s-го інтервалу

$$\psi_s''(x) = 2\frac{y_{s-2} - 2y_{s-1} + y_s}{2h^2} + 6\frac{-y_{s-2} + 3y_{s-1} - 3y_s + y_{s+1}}{6h^3}h$$

або після спрощення

$$\psi_s''(x) = \frac{y_{s-1} - 2y_s + y_{s+1}}{h^2}.$$

На початку (s+1)-го інтервалу

$$\psi_{s+1}''(x_s) = 2a_{2,s+1}$$

або, що те ж саме згідно з при заміні s на s+1

$$\psi''_{s+1}(x)=\frac{y_{s-1}-2y_s+y_{s+1}}{h^2}=2a_{2,s+1}.$$

Порівняння та підтверджує неперервність другої похідної на стику s-го та (s+1)-го інтервалів.

Третя похідна

$$\psi'''_s(x)=6a_{3,s}.$$

На кінці s-го інтервалу

$$\psi'''_s(x_s)=6a_{3,s}$$

або після підстановки

$$\psi'''_s(x_s)=\frac{-y_{s-2}+3y_{s-1}-3y_s+y_{s+1}}{h^3}.$$

На початку (s+1)-го інтервалу

$$\psi'''_{s+1}(x_s)=6a_{3,s+1}$$

або, з врахуванням ,

$$\Psi'''_{s+1}(x_s)=\frac{-y_{s-1}+3y_s-3y_{s+1}+y_{s+2}}{h^3}.$$

Порівняння з показує, що неперервність третьої похідної на стиках інтервалів не гарантується (графік третьої похідної, як це випливає з – це ступінчаста лінія).

Розглянемо тепер визначення коефіцієнтів - на кінцях сплайна.

В точці x_0 згідно з

$$\Psi_1(x_0)=a_{0,1}$$

або після підстановки

$$\Psi_1(x_0)=\frac{y_{-1}+4y_0+y_1}{6}.$$

Але ж y_{-1} у нас немає. Отже його треба придумати. Як? А згідно з граничною умовою на лівому кінці сплайна.

Нехай на лівому кінці має місце гранична умова 0-го роду (*Ngl*=0), а саме-умова проходження сплайнової кривої через точку (x_0,y_0). Тоді

$$y_0 = \frac{y_{-1} + 4y_0 + y_1}{6},$$

звідки

(*Ngl*=0) $y_{-1} = 2y_0 - y_1.$

Черговий варіант-на лівому кінці задано значення першої похідної y_0', тобто гранична умова 1-го роду (*Ngl*=1).

Згідно з (12) при *s*=0

$$\frac{y_1 - y_{-1}}{2h} = y_0',$$

звідки

(*Ngl*=1) $y_{-1} = y_1 - 2hy_0'.$

Наступний варіант-на лівому кінці задано значення другої похідної y_0'' (*Ngl*=2-гранична умова 2-го роду). Згідно з при *s*=0

$$\frac{y_{-1} - 2y_0 + y_1}{h} = y''_0,$$

звідки

(*Ngl*=2) $y_{-1} = 2y_0 - y_1 + h^2 y_0''.$

Гранична умова 3-го роду (*Ngl*=3) гарантує підвищену гладкість сплайну на його початковій ділянці (третя похідна на стику 1-го та 2-го інтервалів не терпить розриву), тобто згідно

$a_{3,1} = a_{3,2}$

або після підстановки

$$-y_{-1} + 3y_0 - 3y_1 + y_2 = -y_0 + 3y_1 - 3y_2 + y_3,$$

звідки

(*Ngl*=3) $y_{-1} = 4y_0 - 6y_1 + 4y_2 - y_3.$

При граничних умовах 4-го роду (*Ngl*=4) вимагають, щоб виконувалась умова періодичності, тобто значення самої функції та її першої та другої похідних у точках x_0 та x_m відповідно співпадали. Отже,

$$\Psi_1(x_0) = \Psi_m(x_m)$$

або

$$a_{0,1} = a_{0,m} + a_{1,m}h + a_{2,m}h^2 + a_{3,m}h^3.$$

Підставляємо значення коефіцієнтів

$$\frac{y_{-1} + 4y_0 + y_1}{6} = \frac{y_{m-2} + 4y_{m-1} + y_m}{6} + \frac{y_m - y_{m-2}}{2h}h + \frac{y_{m-2} - 2y_{m-1} + y_m}{2h^2}h^2 + \\ + \frac{-y_{m-2} + 3y_{m-1} - 3y_m + y_{m+1}}{6h^3}h^3$$

або після спрощення

$$y_{-1} + 4y_0 + y_1 = y_{m-1} + 4y_m + y_{m+1}.$$

Умова неперервності перших похідних

$$\Psi_1'(x_0) = \Psi_m'(x_m)$$

або

$$a_{1,1} = a_{1,m} + 2a_{2,m}h + 3a_{3,m}h^2.$$

І, нарешті, після підстановки значень коефіцієнтів

$$\frac{y_1 - y_{-1}}{2h} = \frac{y_m - y_{m-2}}{2h} + 2\frac{y_{m-2} - 2y_{m-1} + y}{2h^2}h + 3\frac{-y_m + 3y_{m-1} - 3y_m + y_{m+1}}{6h^3}h^2$$

Спростимо цей вираз

$$-y_{-1} + y_1 = -y_{m-1} + y_{m+1}.$$

Умова неперервності другої похідної

$$\Psi_1''(x_0) = \Psi_m''(x_m)$$

або

$$2\frac{y_{-1} - 2y_0 + y_1}{2h^2} = 2\frac{y_{m-2} - 2y_{m-1} + y_m}{2h^2} + 6\frac{-y_{m-2} + 3y_{m-1} - 3y_m + y_{m+1}}{6h^3}h.$$

Спрощуємо

$$y_{-1} - 2y_0 + y_1 = y_{m-1} - 2y_m + y_{m+1}.$$

Розглядаємо тепер - як систему рівнянь.

$$y_{-1} + 4y_0 + y_1 = y_{m-1} + 4y_m + y_{m+1};$$

$$-y_{-1} + y_1 = -y_{m-1} + y_{m+1};$$

$$y_{-1} + 2y_0 + y_1 = -y_{m-1} - 2y_m + y_{m+1}.$$

Вираховуємо з

$$6y_0 = 6y_m,$$

звідки

(*Ngr*=*Ngl*=4) $y_0 = y_m$

Це очевидно - умова неперервності періодичної функції.

Вираховуємо з

$$2y_{-1} + 4y_0 = 2y_{m-1} + 4y_m,$$

звідки

(*Ngr*=*Ngl*=4) $y_{-1} = y_{m-1} + 2y_m - 2y_0.$

Додаємо та

$$4y_0 + 2y_1 = 4y_m + 2y_{m+1},$$

звідки

(*Ngr*=*Ngl*=4) $y_{m+1} = 2y_0 + y_1 - 2y_m.$

Розглянемо тепер варіанти граничних умов на правому кінці згладжуючого сплайна.

(*Ngr*=0); $\psi_m(x_m) = y_m.$

Підставляємо замість $\psi_m(x_m)$ формулу при $s = m,\ x = x_m$

$$a_{0,m} + a_{1,m}h_m + a_{2,m}h_m^{\,2} + a_{3,m}h_m^{\,3} = y_m,$$

з урахуванням - маємо:

$$\frac{y_{m-2} + 4y_{m-1} + y_m}{6} + \frac{y_m - y_{m-2}}{2} + \frac{y_{m-2} - 2y_{m-1} + y_m}{2} + \frac{-y_{m-2} + 3y_{m-1} - 3y_m + y_{m+1}}{6} = y_m$$

або після спрощення:

$$y_{m+1} = 2y_m - y_{m-1};\quad (N_{gr} = 0)$$

(Ngr=1); $\psi_m'(x_m) = y_m'.$

Підставляємо замість $\psi'_m(x_m)$ його значення з при $s=m$, $x=x_m$

$$a_{1,m}+2a_{2,m}h_m+3a_{3,m}h_m^{\,2}=y_m'.$$

Підставляємо значення коефіцієнтів з - при $s=m$

$$\frac{y_m-y_{m-2}}{2h}+\frac{y_{m-2}-2y_{m-1}+y_m}{h}+\frac{-y_{m-2}+3y_{m-1}-3y_m+y_{m+1}}{2h}=y_m',$$

після спрощення одержуємо:

(Ngr=1); $y_{m+1}=y_{m-1}+2hy_m'$;

(Ngr=2); $\psi_m''(x_m)=y_m''$,

використовуємо при $S=m$, $x=x_m$

$$2a_{2,m}+6a_{3,m}h_m=y_m'',$$

підставляємо значення коефіцієнтів з , при $s=m$

$$\frac{y_{m-2}-2y_{m-1}+y_m}{h^2}+\frac{-y_{m-2}+3y_{m-1}-3y_m+y_{m+1}}{h^2}=y_m'',$$

після спрощення:

$$y_{m+1}=2y_m-y_{m-1}+h^2y_m''; \qquad (N_{gr}=2)$$

(Ngr=3); $\psi_{m-1}'''(x_{m-1})=\psi_m'''(x_{m-1})$.

Згідно з $6a_{3,m-1}=6a_{3,m}$,

або після скорочення $a_{3,m-1}=a_{3,m}$.

Підставляємо значення коефіцієнтів з (3/45)

$$-y_{m-3}+3y_{m-2}-3y_{m-1}+y_m=-y_{m-2}+3y_{m-1}-3y_m+y_{m+1},$$

звідки

(Ngr=2); $y_{m+1}=4y_m-6y_{m-1}+4y_{m-2}-y_{m-3}$;

Описаний алгоритм реалізується для згладжування масиву в підпрограмі *SplSgl*.

```
procedure SplSgl(Y:Coefr;Ngl,Ngr:integer;D0,Dm:real;var
Ys:Coefr) ;
var m,s:integer;
    H,H2:real;
```

```
begin
  m := round (Y[-1]); Ys[-1]:= m; H:=1; H2:= sqr (H);
  if (Ngl=4) and (Ngl=4)
    then
      begin
        Y[-1]:=Y[m-1]+2*Y[m]+2*Y[0];
        Y[m+1]:= 2*Y[0]+Y[1] -2*Y[m]
      end
    else
      begin
        case Ngl of
          0: Y[-1]:= 2*Y[0]-Y[1];
          1: Y[-1]:= Y[1]-2*H*D0;
          2: Y[-1]:= 2*Y[0]-Y[-1]+H2*D0;
          3:Y[-1]:= 4*Y[0]-6*Y[1]+4*Y[2]-Y[3]
        end;
        case Ngl of
          0: Y[m+1]:= 2*Y[m]-Y[m-1];
          1: Y[m+1]:= Y[m-1]+2*H*Dm;
          2: Y[m+1]:= 2*Y[m]-Y[m-1]+H2*Dm;
          3: Y[m+1]:= 4*Y[m]-6*Y[m-1]+4*Y[m-2]-Y[m-3]
        end;
    end;
    for s:= 0 to m do
      Ys[s]:= (Y[s-1]+4*Y[s]+Y[s+1])/6
end;
```

Роботу методе продемонстровано в програмі DemSpl, яка дозволяє проводити згладжування попередньо визначених сплайнів. Це продемонстровано на рис.3.3, де а) – це кубічний сплайн, що згладжує задані точки, б) та в) - сплайни, які згладжують криві а) та б), відповідно.

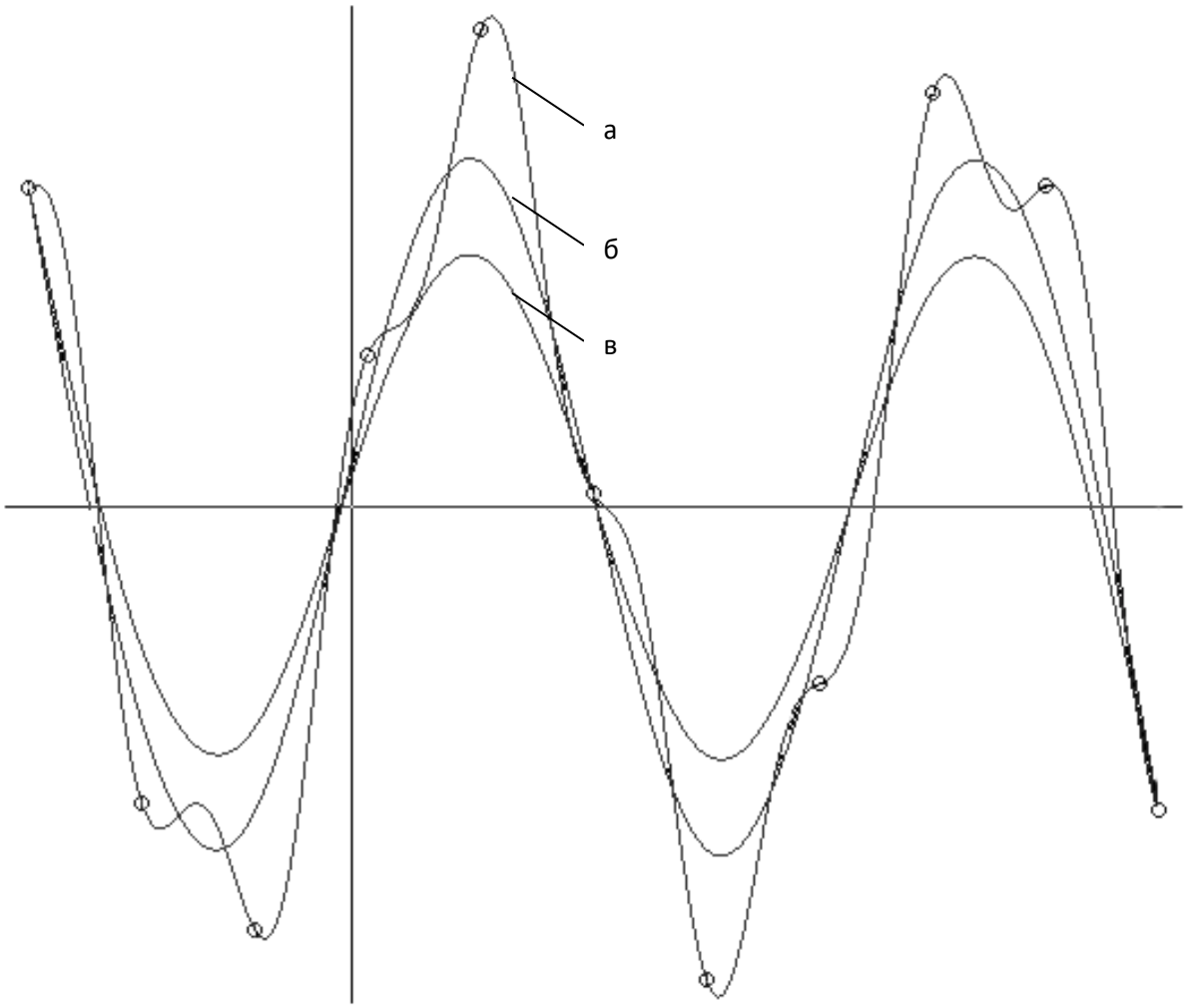

Рис. 7. Послідовне згладжування кубічними сплайнами

а) згладжувана крива; б) перше згладжування; в) друге згладжування

Лекція 8. Інтерполяційні кубічні B-сплайни

Нехай, як і в попередньому параграфі задана множина значень аргументу x (вузлів апроксимації)

$$x_0<x_1<x_2<..<x_{m-1}<x_m$$

зі сталим кроком $h_s=x_s-x_{s-1}$=const.

Кожному значенню x із вказаної множини поставлено у відповідність значення функції

$$y_0,y_1,y_2,..,y_{m-1},y_m.$$

Кубічний B-сплайн (базовий сплайн) для інтервалу $[x_{s-2}..x_{s+2}]$ визначається таким чином

$$B_s(x)=\begin{cases}\frac{1}{6}(u_s+2)^3, & x\in[x_{s-2},x_{s-1}],\\ \frac{2}{3}-\frac{1}{2}\left(u_s^3+2u_s^2\right), & x\in[x_{s-1},x_s],\\ \frac{2}{3}+\frac{1}{2}\left(u_s^3-2u_s^2\right), & x\in[x_s,x_{s+1}],\\ \frac{1}{6}(2-u_s)^3, & x\in[x_{s+1},x_{s+2}],\\ 0 \text{ для інших } x,\end{cases}$$

де $u_s=\dfrac{x-x_s}{h}$.

Обчислимо значення першої, другої та третьої похідних від сплайну за x.

$$B_s^{'}(x)=\begin{cases}\frac{1}{2h}(u_s+2)^2, & x\in[x_{s-2},x_{s-1}],\\ -\frac{1}{2h}\left(3u_s^2+4u_s\right), & x\in[x_{s-1},x_s],\\ \frac{1}{2h}\left(3u_s^2-4u_s\right), & x\in[x_s,x_{s+1}],\\ -\frac{1}{2h}(2-u_s)^2, & x\in[x_{s+1},x_{s+2}],\\ 0 \text{ для інших } x.\end{cases}$$

$$B_s^{''}(x)=\begin{cases}\frac{1}{h^2}(u_s+2), & x\in[x_{s-2},x_{s-1}],\\ -\frac{1}{2h^2}\left(6u_s+4\right)=-\frac{1}{h^2}(3u_s+2), & x\in[x_{s-1},x_s],\\ \frac{1}{2h^2}\left(6u_s-4\right)=\frac{1}{h^2}(3u_s-2), & x\in[x_s,x_{s+1}],\\ \frac{1}{h^2}(2-u_s), & x\in[x_{s+1},x_{s+2}],\\ 0 \text{ для інших } x.\end{cases}$$

$$B'''_s(x)=\begin{cases}\dfrac{1}{h^3}, & x\in[x_{s-2},x_{s-1}],\\ -\dfrac{1}{3h^3}, & x\in[x_{s-1},x_s],\\ \dfrac{1}{3h^3}, & x\in[x_s,x_{s+1}],\\ -\dfrac{1}{h^3}, & x\in[x_{s+1},x_{s+2}],\\ 0 \text{ для інших } x.\end{cases}$$

Графічно кубічні В-сплайни та їх похідні мають вигляд як на рис.3.4.

Відрізок $[x_{s-2}, x_{s+2}]$ називається носієм функції $Bs(x)$.

Доповнюємо сітку $x_0<x_1<x_2<..<x_m$, де $x_0=a$, $x_m=b$, допоміжними вузлами $x_{-3}<x_{-2}<x_{-1}<a$ та $b<x_{m+1}<x_{m+2}<x_{m+3}$.

За розширенною сіткою

$x_{-3}<x_{-2}<x_{-1}<a<x_1<x_2<..<x_{m-1}<b<x_{m+1}<x_{m+2}<x_{m+3}$

можна побудувати сімейство з $(m+3)$ кубічних В-сплайнів типу $Bs(x)$, $s=-1,0,1,..,m-1,m,m+1$.

Це сімейство утворює базис в просторі кубічних сплайнів на відрізку $[a,b]$. Тим самим, довільний кубічний сплайн $\psi(x)$, побудований на відрізку $[a,b]$ за сіткою з $(m+1)$ вузлів, може бути представлений на цьому відрізку у вигляді лінійної комбінації

$$\psi(x)=\sum_{s=-1}^{m+1} b_s B_s(x).$$

Отже, задача формування В-сплайна зводиться до визначення b_s, $-1\le s\le m+1$. В-сплайн повинен задовольняти трьом вимогам:

бути неперервним від x та забезпечувати рівність значень В-сплайна відповідним значенням y_s у вузлах інтерполяції;

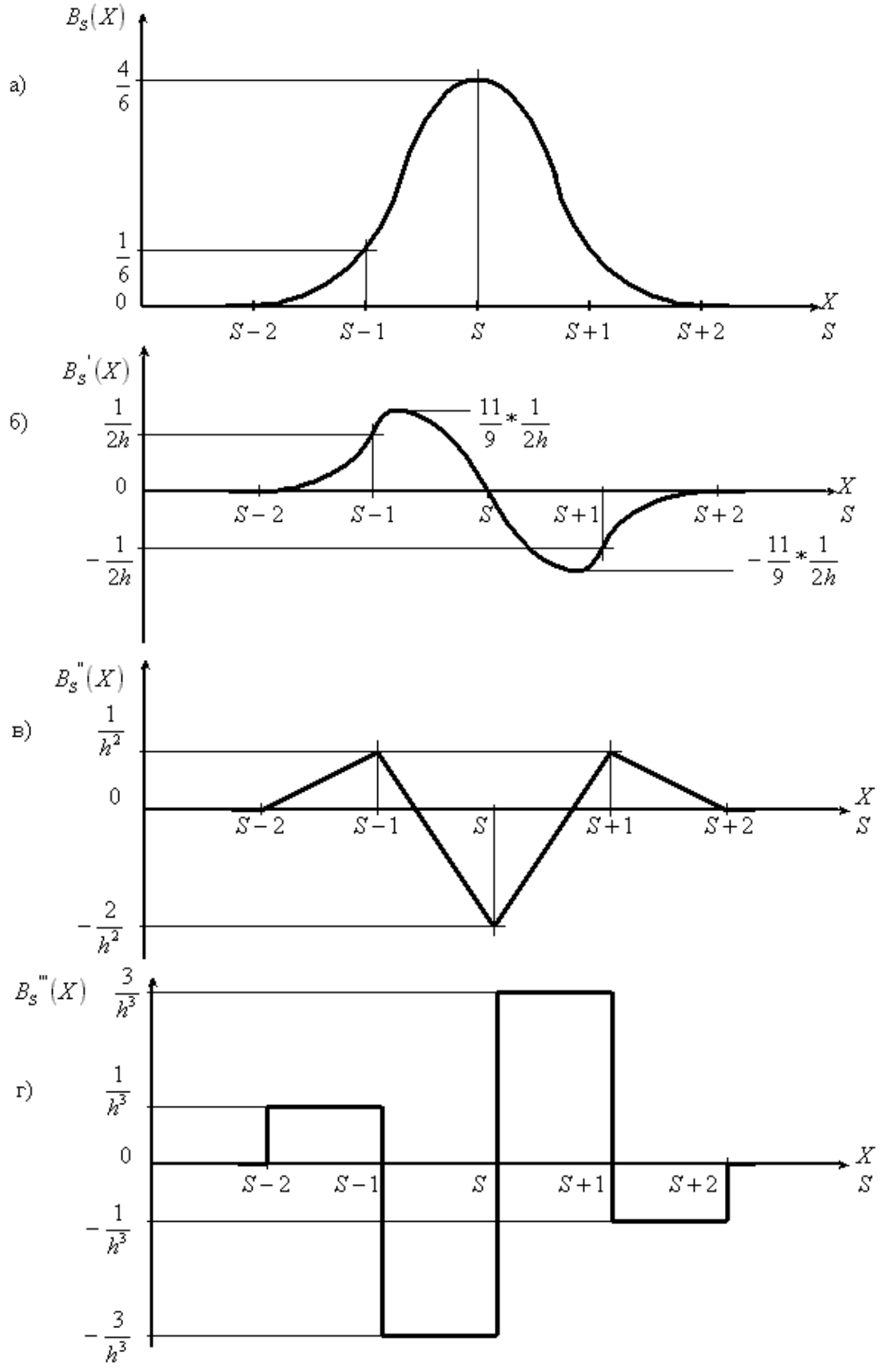

Рис.8. Графіки В-сплайна та його похідних

2)неперервною на відрізку [a,b] має бути перша похідна від В-сплайна;

3)неперервною на відрізку [a,b] має бути друга похідна від В-сплайна.

Перевіримо спочатку неперервність окремо взятого елемента сплайна, тобто $B_s(x)$.

Точка $x = x_{S-2}$, $u_S = -2$.

Ліва границя $B_S(x_{S-2}) = 0$ (0 в усіх інших точках) .

Права границя $B_s(x_{s-1}) = \frac{1}{6}(u_s + 2)\Big|_{U_S=-2} = 0$.

Отже в точці $x = x_{S-2}$ В- сплайн є неперервним .

Точка $x = x_{S-1}$, $u_S = -1$.

Ліва границя $B_s(x_{s-1}) = \frac{1}{6}(u_s + 2)\Big|_{U_S=-1} = \frac{1}{6}$.

Права границя $B_S(x_{S-1}) = \frac{2}{3} - \frac{1}{2}u_S{}^2(u_S + 2)\Big|_{U_S=-1} = \frac{2}{3} - \frac{1}{2} = \frac{1}{6}$.

Виходить , що і в точці x_{S-1} В- сплайн є неперервним .

Точка $x = x_S$, $u_S = 0$.

Ліва границя $B_S(x_S) = \frac{2}{3} - \frac{1}{2}u_S{}^2(u_S + 2)\Big|_{U_S=0} = \frac{2}{3} = \frac{4}{6}$.

Права границя $B_S(x_S) = \frac{2}{3} + \frac{1}{2}u_S{}^2(u_S - 2)\Big|_{U_S=0} = \frac{2}{3} = \frac{4}{6}$.

І в точці x_S неперервність має місце .

Точка $x = x_{S+1}$, $u_S = 1$.

Ліва границя $B_S(x_{S+1}) = \frac{2}{3} + \frac{1}{2}u_S{}^2(u_S - 2)\Big|_{U_S=1} = \frac{2}{3} - \frac{1}{2} = \frac{1}{6}$.

Права границя $B_S(x_{S+1}) = \frac{1}{6}(2 - u_S)^3\Big|_{U_S=1} = \frac{1}{6}$.

Точка $x = x_{S+2}$, $u_S = 2$.

Ліва границя $B_S(x_{S+2}) = \frac{1}{6}(2 - u_S)^3\Big|_{U_S=2} = 0$.

Права границя $B_S(x_{S+2}) = 0$ (в усіх інших точках)

Таким чином , ми пересвідчились , що В- сплайн є неперервною функцією на всьому відрізку його існування .

Перевіримо тепер неперервність першої похідної від B- сплайна .

Точка $x = x_{S-2}$, $u_S = -2$.

Ліва границя $B_S{}'(x_{S-2}) = 0$ (в усіх інших точках).

Права границя $B_S{}'(x_{S-2}) = \frac{1}{2h}(u_s + 2)^2\Big|_{U_S=-2} = 0$.

Висновок: в точці x_{S-2}, $B_S{}'(x)$ є неперервною функцією .

Точка $x = x_{S-1}$, $u_S = -1$.

Ліва границя $B_S{}'(x_{S-1}) = \frac{1}{2h}(u_s + 2)^2\Big|_{U_S=-1} = \frac{1}{2h}$.

Права границя $B_S{}'(x_{S-1}) = \frac{1}{2h}(3u_s^2 + 4u_s)\Big|_{U_S=-1} = \frac{1}{2h}$.

В точці x_{S-2} перша похідна від B- сплайна – неперервна .

Точка $x = x_S$, $u_S = 0$.

Ліва границя $B_S{}'(x_S) = -\frac{1}{2h}(3u_s^2 + 4u_s)\Big|_{U_S=0} = 0$.

Права границя $B_s'(x_s) = \frac{1}{2h}(3u_s^2 + 4u_s)\Big|_{U_S=0} = 0$.

В точці $x = x_s$ функція $B_S{}'(x_S)$ неперервна .

Точка $x = x_{S+1}$, $u_S = 1$.

Ліва границя $B_S{}'(x_{S+1}) = \frac{1}{2h}\left(3u_S{}^2 - 4u_S\right)\Big|_{U_S=1} = -\frac{1}{2}$.

Права границя $B_S{}'(x_{S+1}) = -\frac{1}{2h}(2 - u_S)^2\Big|_{U_S=1} = -\frac{1}{2h}$.

Неперервність $B_S{}'(x)$ в точці x_{S+1} має місце .

Точка $x = x_{S+2}$, $u_S = 2$.

Ліва границя $B_S{}'(x_{S+2}) = -\frac{1}{2h}(2 - u_S)^2\Big|_{U_S=2} = 0$.

Права границя $B_S{}'(x_{S+2}) = 0$ (в усіх інших точках).

Отже, і в точці x_{S+2} спостерігаємо неперервність функції $B'(x)$.

Перевіримо тепер неперервність другої похідної від B-сплайну .

Точка $x = x_{S-2}$, $u_S = -2$.

Ліва границя $B_S''(x_{S-2}) = 0$ (в усіх інших точках) .

Права границя $B_s''(x_{s-2}) = \frac{1}{h^2}(u_s + 2)\Big|_{U_S=-2} = 0$.

Друга похідна в точці x_{S-2} неперервна .

Точка $x = x_{S-1}$, $u_S = -1$.

Ліва границя $B_S''(x_{S-2}) = \frac{1}{h^2}(u_s + 2)\Big|_{U_S=-1} = \frac{1}{h^2}$.

Права границя $B_S''(x_{S-1}) = -\frac{1}{2h^2}(6u_s + 4)\Big|_{U_S=-1} = \frac{1}{h^2}$.

Друга похідна в точці x_{S-1} неперервна .

Точка $x = x_S$, $u_S = 0$.

Ліва границя $B_S''(x_S) = -\frac{1}{2h^2}(6u_s + 4)\Big|_{U_S=0} = -\frac{2}{h^2}$.

Права границя $B_S''(x_S) = -\frac{1}{2h^2}(6u_s - 4)\Big|_{U_S=0} = -\frac{2}{h^2}$.

Отже, і в точці x_S маємо неперервну другу похідну від B-сплайну.

Точка $x = x_{S+1}$, $u_S = 1$.

Ліва границя $B_S''(x_{S-1}) = \frac{1}{2h^2}(6u_s - 4)\Big|_{U_S=1} = \frac{1}{h^2}$.

Права границя $B_S''(x_{S-1}) = \frac{1}{h^2}(2 - u_s)\Big|_{U_S=1} = \frac{1}{h^2}$.

І в точці $x = x_{S+1}$ друга похідна неперервна .

Точка $x = x_{S+2}$, $u_S = 2$.

Ліва границя $B_S''(x_{S+2}) = \frac{1}{h^2}(2 - u_S)\Big|_{U_S=2} = 0$.

Права границя $B_S''(x_{S+2}) = 0$ (в усіх інших точках)

На кінці інтервалу (в точці x_{S+2}) друга похідна неперервна.

Таким чином показано, що на всьому інтервалі існування сам В- сплайн та його перша та друга похідні є неперервними функціями. Неперервність цих функцій в межах будь- якого підінтервалу гарантується тим , що поліном є неперервною функцію свого аргументу . Третя похідна від В- сплайну, як це випливає з формули , не є неперервною функцією –має розриви на стиках підінтервалів.

Умова проходження В-сплайну $\Psi(x)$ через точку (x_s,y_s) на площині *XOY*

$$b_{s-1}B_{s-1}(x_s)+b_sB_s(x_s)+b_{s+1}B_{s+1}(x_s)=y_s,\ 0\le x\le m.$$

Користуючись формулами , визначимо $B_{s-1}(x_s)$, $B_s(x_s)$, та $B_{s+1}(x_s)$.

$$B_{s-1}(x_s)=\frac{1}{6}(2-1)^3=\frac{1}{6},$$

$$B_s(x_s)=\frac{2}{3}-\frac{1}{2}(0^3+2\cdot 0^2)=\frac{2}{3},$$

$$B_{s+1}(x_s)=\frac{1}{6}(-1+2)^3=\frac{1}{6}.$$

Отже, набуває вигляду

$$b_{s-1}+4b_s+b_{s+1}=6y_s,\ 0\le s\le m.$$

Це система з (m+1) рівнянь відносно (m+3) невідомих b_s -1≤s≤m+1.

Її треба доповнити двома рівняннями граничних умов – на лівому та на правому кінцях В-сплайну. Нехай, на лівому кінці маємо граничну умову 1-го роду, тобто задано значення першої похідної y_0' в точці x_0 (*Ngl*=1).

Диференціюємо при s=0

$$b_{-1}B'_{-1}(x_0)+b_0B'_0(x_0)+b_1B'_1(x_0)=y'_0,\quad 0\le x\le m,$$

Користуючись формулами , визначаємо $B'_{-1}(x_0), B'_0(x_0)$ та $B'_1(x_0)$

$$B'_{-1}(x_0)=-\frac{1}{2h}(2-1)^2=-\frac{1}{2h},$$

$$B'_0(x_0)=-\frac{1}{2h}(3\cdot 0^2+4\cdot 0\ )=0,$$

$$B'_1(x_0)=\frac{1}{2h}(-1+2)^2=\frac{1}{2h}.$$

І тоді набуває вигляду

(*Ngl*=1) $\quad -b_{-1}+b_1=2hy_0^{'}$

Коли ж на лівому кінці B-сплайну має місце гранична умова 2-го роду (*Ngl*=2), то, диференціюючи двічі при *s*=0

$b_{-1}B_{-1}^{''}(x_0)+b_0B_0^{''}(x_0)+b_1B_1^{''}(x_0)=y_0^{''}.$

Визначимо $B_{-1}^{''}(x_0), B_0^{''}(x_0)$ та $B_1^{''}(x_0)$ з

$$B''_{-1}(x_0)=\frac{1}{h^2}(-1+2) \ =\frac{1}{h^2},$$

$$B''_{0}(x_0)=-\frac{1}{h^2}(3\cdot 0 \ +2)=-\frac{2}{h^2},$$

$$B''_{1}(x_0)=\frac{1}{h^2}(-1+2) \ =\frac{1}{h^2}.$$

Отже, набуває вигляду

(*Ngl*=2) $\quad b_{-1}-2b_0+b_1=y_0^{''}h^2.$

А тепер звернемо увагу на те, що система є тридіагональною. Будемо розв'язувати її методом прогонки, послідовно виключаючи з кожного з рівнянь b_{s-1}, тобто приводячи чергове (*s*-те) рівняння до вигляду

$b_s=C_sb_{s+1}+D_s, \ \ 0\le s\le m.$

Підставимо в замість b_{s-1}

$C_{s-1}b_s+D_{s-1}+4b_s+b_{s+1}=6y_s.$

Зводимо подібні та розв'язуємо відносно b_s

$$b_s=\frac{-1}{C_{s-1}+4}b_{s+1}+\frac{6y_s-D_{s-1}}{C_{s-1}+4}.$$

Порівнюючи з , бачимо, що

$$\begin{cases} C_s=\dfrac{-1}{C_{s-1}+4}, \ \ 0\le s\le m+1, \\ D_s=\dfrac{6y_s-D_{s-1}}{C_{s-1}+4}, \ 0\le s\le m+1. \end{cases}$$

Формули , -рекурентні, щоб вони "запрацювали", визначимо з "лівої" граничної умови C_0 та D_0.

Рівняння при s=0 має вигляд

$$b_{-1} + 4b_0 + b_1 = 6y_0$$

При Ngl=1 рівняння утворює з систему

$$\begin{cases} b_{-1} + 4b_0 + b_1 = 6y_0, \\ -b_{-1} + b_1 = 2hy_0'. \end{cases}$$

Додаємо рівняння цієї системи

$$4b_0 + 2b_1 = 6y_0 + 2y_0'h$$

або

$$b_0 = -0{,}5b_1 + \frac{3y_0 + hy_0'}{2}$$

Порівнюємо з при s=0

$$(Ngl=1) \qquad \begin{cases} C_0 = -0{,}5, \\ D_0 = \dfrac{3y_0 + hy_0'}{2}. \end{cases}$$

При Ngl=2 систему утворюють рівняння та

$$\begin{cases} b_{-1} + 4b_0 + b_1 = 6y_0, \\ b_{-1} - 2b_0 + b_1 = y_0''h^2. \end{cases}$$

Віднімаємо друге рівняння від першого

$$6b_0 = 6y_0 - y_0''h^2$$

або

$$b_0 = 0 \cdot b_1 + y_0 - y_0'' \frac{h^2}{6}$$

Порівнюємо з при s=0, звідки

$$(Ngl=2) \qquad \begin{cases} C_0 = 0, \\ D_0 = y_0 - y_0'' \dfrac{h^2}{6}. \end{cases}$$

При s=m-1 набуває вигляду

$$b_{m-1} = C_{m-1}b_m + D_{m-1}.$$

Гранична умова I роду на правому кінці B-сплайну

$$b_{m-1}B'_{m-1}(x_m) + b_m B'_m(x_m) + b_{m+1}B'_{m+1}(x_m) = y'_m.$$

Легко бачити з , що

$$B'_{m-1}(x_m) = -\frac{1}{2h},$$

$$B'_m(x_m) = 0,$$

$$B'_{m+1}(x_m) = \frac{1}{2h},$$

звідки набуває вигляду

$$-b_{m-1} + b_{m+1} = 2hy'_m.$$

З іншого боку з при $s=m$ маємо

$$b_{m-1} + 4b_m + b_{m+1} = 6y_m.$$

Якщо з відняти , то

$$2b_{m-1} + 4b_m = 6y_m - 2hy'_0.$$

або, ділячи на 2,

$$b_{m-1} + 2b_m = 3y_m - hy'_0.$$

Віднімаємо тепер від

$$0 = (C_{m-1} - 2)b_m + D_{m-1} - 3y_m + hy'_m,$$

звідки

(Ngr=1) $$b_m = \frac{D_{m-1} - 3y_m + hy'_m}{2 - C_{m-1}}.$$

При Ngr=2 гранична умова на правому кінці B-сплайна

$$b_{m-1}B''_{m-1}(x_m) + b_m B''_m(x_m) + b_{m+1}B''_{m+1}(x_{m+1}) = y''_m.$$

Скористаємось формулами для визначення других похідних у

$$B''_{m-1}(x_m) = \frac{1}{2}(2-1) = \frac{1}{h^2},$$

$$B''_m(x_m) = \frac{1}{2h^2}(6\cdot 0 - 4) = \frac{-2}{h^2},$$

$$B''_{m+1}(x_m) = \frac{1}{h^2}(-1+2) = \frac{1}{h^2}.$$

Отже, набуває вигляду

$$b_{m-1}\frac{1}{h^2}-b_m\frac{2}{h^2}+b_{m+1}\frac{1}{h^2}=y''_m$$

або

$$b_{m-1}-2b_m+b_{m+1}=y''_0h^2.$$

Віднімаємо з

$$6b_m=6y_m-y''_mh^2,$$

звідки

(*Ngr*=2) $b_m=y_m-y''_m\frac{h^2}{6}$.

Зворотній хід виконується за формулою , в якій *s* змінюється від *m*-1 до 0. Значення b_{m+1} тоді можна обчислити за формулою, що одержується з

$$b_{m+1}=6y_m-b_{m-1}-4b_m.$$

Аналогічно з формули

$$b_{-1}=6y_0-b_1-4b_0.$$

Підпрограма *Bspline*, яка формує масив *B* коефіцієнтів b_s ($-1\le s\le m+1$) може мати такий вигляд при нормованому *x*, що змінюється в діапазоні $x\in[0,m]$.

```
procedure BSpline (Y:Coefr; Ngl,Ngr:integer;D0,Dm:real;
var m:integer; var B:Coefr);
var s:integer;
    z,h:real;
    A,C:Coefr;
begin
  m:=round(Y[-1]); h:=1;
  case Ngl of
    1: begin A[0]:=-0.5; C[0]:=(3*Y[0]+h*D0)/2 end;
    2:begin A[0]:=0; C[0]:=Y[0]-D0*sqr(h)/6 end
  end;
  for s:=1 to m-1 do
    begin
      z:=A[s-1]+4;
```

```
        A[s]:=-1/z; C[s]:=(6*Y[s]-C[s-1])/z
    end;
  case Ngr of
    1: B[m]:=(-C[m-1]+3*Y[m]-h*Dm)/(2+A[m-1]);
    2: B[m]:=Y[m]-Dm*sqr(h)/6
  end;
  for s:=m-1 downto 0 do
    B[s]:=A[s]*B[s+1]+C[s];
  B[-1]:=6*Y[0]-B[1]-4*B[0];
  B[m+1]:=6*Y[m]-B[m-1]-4*B[m]
end;
```

Підпрограма-функція *BSplint* для нормованого діапазону $x(0 \le x \le m) \in [0,m]$буде виглядатиме наступним чином

```
function BSplint(x:real):real;
var s:integer;
 begin
  s:=trunc(x); x:=frac(x);
  if s>m
    then  BSplint:=(B[m-1]+4*B[m]+B[m+1])/6
    else
      BSplint:= B[s-1]*(1-x)*sqr(1-x)/6+
                B[s]*(2/3+0.5*sqr(x)*(x-2))+
                B[s+1]*(2/3-0.5*sqr(x-1)*(x+1))+
                B[s+2]*x*sqr(x)/6
 end;
```

Порівняння звичайного інтерполяційного кубічного сплайну та інтерполяційного B-сплайну. Звичайний інтерполяційний сплайн є сукупністю кубічних поліномів для кожного з підінтервалів. Інтерполяційний B-сплайн в кожній точці (на кожному з підінтервалів) є сумою кубічних

поліномів, отже, є сам кубічним поліномом. Оскільки при заданих граничних умовах, як звичайний, так і В-сплайни визначаються однозначно, то очевидно, що на кожному з підінтервалів ми будемо мати один і той же кубічний поліном, тобто звичайний сплайн і В-сплайн в математичному плані є повністю еквівалентними. Отже, при потребі можна перераховувати один в інший і навпаки.

Для звичайного кубічного інтерполяційного сплайна функція *Splint* як було показано раніше, має вигляд

```
function Splint(x:real):real;
var s:integer;
begin
  s:=trunc(x)+1;  x:=frac(x);
  Splint:=((A3[s]*x+A2[s])*x+A1[s])*x+A0[s]
end;
```

Для В-сплайну аналогічно

```
function BSplint(x:real):real;
var s:integer;
begin
  s:=trunc(x)+1;  x:=frac(x)
  BSplint:=B[s-1]*sqr(1-x)*(1-
x)/6+B[s]*(2/3+0.5*sqr(x)*
            (x-2))+B[s+1]*(2/3-0.5*sqr(x1)*(x+1))+
            B[s+2]*sqr(x)*x/6
end;
```

В цих двох підпрограмах x вважається нормованим ($0 \le x \le m$) і, крім того, приймається, що відповідні звичайні та В-сплайни є інтерполяційними та сформовані при однакових граничних умовах.

Розкриємо дужки та зведемо подібні в формулі для *Bsplint*

Bsplint:=x^3{*-B[s-1]/6+B[s]/2-B[s+1]/2+B[s+2]/6}+*x*^{2}{*B[s-1]/2-B[s]+B[s+1]/2}+x{-B[s-1]/2+B[s+1]/2}+{B[s-1]/6+B[s]*2/3+B[s+1]/6}*

Порівнюючи цей вираз з аналогічним виразом з *Splint*, приходимо до висновку, що

*A*3[*s*+1]=(-*B*[*s*-1]+3·*B*[*s*]-3·*B*[*s*+1]+*B*[*s*+1])/6*;*

*A*2[*s*+1]=(*B*[*s*-1]-2·*B*[*s*]+ *B*[*s*+1])/2*;*

*A*1[*s*+1]=(*B*[*s*+1]-*B*[*s*-1])/2*;*

*A*0[*s*+1]=(*B*[*s*-1]+4·*B*[*s*]+ *B*[*s*+1])/6*;*

Обчислимо 3*A*0[*s*+1]- *A*2[*s*+1]. Матимемо

3*A*0[*s*+1]- *A*2[*s*+1]=3·*B*[*s*],

звідки

B[*s*]= *A*0[*s*+1]- *A*2[*s*+1]/3.

Вираховуємо 4/6 *B*[s] з *A*0[*s*+1]

*A*0[*s*+1]- (4/6) ·*B*[*s*]=(*B*[*s*-1]+ *B*[*s*+1])/6

або це те ж саме

6*A*0[*s*+1]- 4*B*[*s*]=*B*[*s*-1]+ *B*[*s*+1].

З *A*1[*s*+1] можна записати

2*A*1[*s*+1]= *B*[*s*+1]-*B*[*s*-1].

Додаючи та одержимо

B[*s*+1]=3*A*0[*s*+1]-2*B*[*s*]+ *A*1[*s*+1].

З *A*1[*s*+1] одержуємо

B[*s*-1]:= *B*[*s*+1]-2*A*1[*s*+1].

І, нарешті, з *A*3[*s*+1] маємо

B[*s*+2]:=6*A*3[*s*+1]+*B*[*s*-1]-3*B*[*s*]+3*B*[*s*+1].

Формули , -, якщо їх застосувати послідовно, дають можливість перерахувати значення *A*3[*s*+1], *A*2[*s*+1], *A*1[*s*+1], *A*0[*s*+1] в значення *B*[*s*-1], *B*[*s*], *B*[*s*+1], *B*[*s*+2], які визначають значення B-сплайнової апроксимуючої функції на інтервалі між точками X_S та $X_{S+!}$.

Формування B-сплайну (масиву *B*:*Coefr*) за масивами коефіцієнтів *A*0, *A*1, *A*2, *A*3:Coefr звичайного інтерполяційного сплайну можна представити у вигляді підпрограми

```
procedure TranSplB ( A0,A1,A2,A3:Coefr; var m:integer;
                                        var B:Coefr);
var s:integer;
begin
  m:=round(A0[-1]);
  for s:=0 to m-1 do
    B[s]:=A0[s+1]-A2[s+1]/3;
  B[-s]:= B[1]-2*A0[1]);
  B[m]:=3*A0[m]-2*B[m-1]+ A1[m];
  B[m+1]:=6*A3[m]+B[m-2]-3*B[m-1]+3*B[m]
end;
```

Формування ж масивів *A*0, *A*1, *A*2, A3:Coefr за масивом *B*, сформованим під B-сплайн, можна представити процедурою *TranBSpl* у відповідності з формулами .

```
procedureTranBSpl(B:Coefr;m:integer;var
                          A0,A1,A2,A3:Coefr);
var s:integer;
begin
  A0[-1]):=m;
  for s:=0 to m do
    begin
      A3[s+1]:=(-B[s-1]+3*B[s]-3*B[s+1]+B[s+2])/6;
      A2[s+1]:=(B[s-1]-2*B[s]+B[s+1])/2;
      A1[s+1]:=(B[s+1]-B[s-1])/2;
      A0[s+1]:=(B[s-1]+4*B[s]+B[s+1])/6
    end
end;
```

Лекція 9. Ковзаючий інтерполяційний поліном

Щойно розглянуте згладжування з використанням кубічного сплайну забезпечує високу якість згладжування, більше того – дозволяє врахувати граничні умови, але в згладжуючому сплайні не передбачено параметра, який би дозволяв регулювати інтенсивність згладжування.

Згладжування методом найменших квадратів, розглянуте у 3.3, орієнтоване на згладжування усього масиву точок одним-єдиним поліномом *n*-ступеня. Ця умова є надто жорсткою, коли згладжувана функція має відносно складну форму.

Можна запропонувати компромісне рішення, а саме: згладжувати поліномом *n*-ступеня не всю криву, а лише її частину довжиною (2*k*+1) вузлів, причому згладжуваний відрізок даної довжини розглядати як ковзаючий. Найвища якість згладжування досягається для центральної точки, отже, ковзаючий відрізок доцільно формувати так, щоб згладжувана точка знаходилася в його центрі. На краях інтервалу доведеться задовольнитися згладжуючим ефектом відрізка, що упирається в лівий, чи, відповідно, в правий край інтервалу.

Розглянутий алгоритм згладжування масиву функції *My:Coefr* реалізується процедурою *KSglMNK*

```
procedure KSglMNK(My:Coefr;k,n:integer;var Ys:Coefr);
var z,s,m:integer;
    Y,A:Coefr;
    Ms:Mafr;
procedure MNK;
var m,z,s:integer;
    B,C:Coef;
    SumB,PC:real;
begin
  m:= round(Y[-1]);
  for s:=0 to m do B[s]:=1;
```

```
  for z:=0 to 2*n do
    begin
      SumB:=0; PC:=0;
      for s:=0 to m do
        begin
          SumB:=SumB+B[s];
          if z <= n then PC:=PC+B[s]*Y[s];
          if z <2*n then B[s]:=B[s]*s
        end;
      C[z]:=SumB;
      if z <= n then Ms [z+1, n+2]:=PC
    end;
  for z:=1 to n+1 do
    for s:=1 to n+1 do Ms[z,s]:=C[z+s-2];
  Sysfur(n+1, Ms, A); A[-1]:= n;
  for s:=0 to n do A[s]:=A[s+1]
end;
begin
  Y[-1]:=2*k; m:=round(My[-1]);
  Ys[-1]:=m; Ys[m+1]:=My[m+1];
  for s:=0 to 2*k do Y[s]:=My[s];
  MNK;
  for s:=0 to k do Ys[s]:=HorReal(A,S);
    for z:=k+1 to m-k-1 do
      begin
        for s:=-k to k do
          Y[s+k]:=My[z+s];
        MNK; Ys[z]:= HorReal(A,k)
      end;
    for s:=0 to 2*k do
```

```
    Y[s]:=My[m-2*k+s];
  MNK;
  for s:=m-k to m do
    Ys[s]:= HorReal(A, s-m+k)
end;
```

В цій підпрограмі прийнято, що крок за *x* між ординатами дорівнює 1, отже, *x* замінено на *s*. Інтенсивність згладжування можна регулювати значеннями *n* та *k*.

Лекція 10. Ковзаючий інтерполяційний кубічний сплайн та B-сплайн

Класичні методи інтерполяції залежностей, заданих таблично, орінтуються, як правило, на апроксимацію даної залежності в околі точки інтерполяції поліномом 1-го або 2-го степеня (лінійна та квадратична інтерполяція). Гарантуючи неперервність апроксимуючої функції, ці методи не гарантують неперервності першої, а тим більше, другої похідних. Отже, якість апроксимації досягається невисока. Апроксимація таблиці кубічним інтерполяційним сплайном (звичайним чи B-сплайном, що еквівалентно, як було показано вище) гарантує неперервність перших двох похідних, що значно підвищує якість апроксимації. Єдиним недоліком такої апроксимації можна вважати значний обсяг пам'яті, який займають коефіцієнти сплайна. B-сплайн у цьому відношенні економічніший, але ж і масив *B*:*Coefr* треба десь зберігати. Відомо, що форма сплайна в околі *s*-го вузла інтерполяції в першу чергу залежить від значень функції, що апроксимується, розташованих в безпосередній близькості від розгляданого вузла. І чим далі розташовані чергові вузли, тим менше вони впливають на форму сплайна в околі s-го вузла. Отже, можна спробувати розв'язувати задачу інтерполяції з допомогою сплайна, що формується на базі набору (2*k*+1) вузлів, центром якого є s-ий вузол.

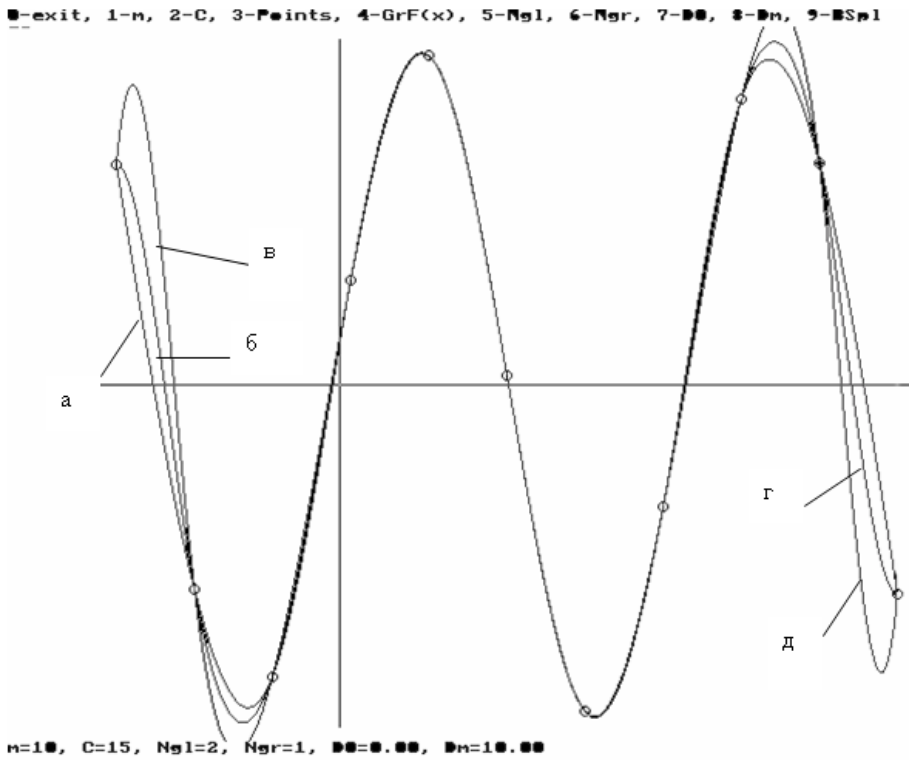

Рис.9. Порівняння результатів апроксимації інтерполяційними B-сплайнами для різних граничних умов:

а) Ngl=2; Ngr=2; D0=0; Dm=0;

б) Ngl=1; Ngr=2; D0=0; Dm=0;

в) Ngl=1; Ngr=2; D0=10; Dm=0;

г) Ngl=2; Ngr=1; D0=0; Dm=0;

д) Ngl=2; Ngr=1; D0=0; Dm=10;

Звичайно, при такому підході виникає проблема врахування граничних умов на кінцях цього ковзного сплайна. У першому наближенні можна орієнтуватись на так званий “природний” варіант практичних умов, коли приймаються граничні умови 2-го роду на обох кінцях з нульовими значеннями других похідних (ненавантажений сплайн). Переходячи від повного сплайну до ковзаючого ми, звичайно, втрачаємо в якості апроксимації, але виграємо в обсязі пам’яті – коефіцієнти того ж таки ковзаючого B-сплайну можна розміщувати не в масиві типу *Coefr*, а в масиві типу *Coef*, якщо *k* не перевищує 15 (а це більше ніж досить). Крім того, масив

B можна розглядати як локальний у відповідній підпрограмі-функції, і тоді пам’ять основної програми взагалі залишається незайманою. Функція ж формування В-сплайна та його використання зосереджуються в одній підпрограмі-функції *BSplintK*, лістинг якої і наводимо.

```
function BsplintK(My:Coefr;k:integer;x:real):real;
var m, s, s1, s2, :integer;
    Y, B, C, D, : Coef; Zn:real;
begin
  m:=round(My[-1]); s:=trunc(x); x:=frac(x);
  s1:=s; s2:=k;
  if s<k then
    begin
      s1:=k; s2:=s
    end;
  if s>m-k then
    begin
      s1:=m-k; s2:=s-m+2*k
    end;
  for s:=-k to k do Y[s+k]:=My[s+s1]);
  m:=2*k; C[0]:=0; D[0]:=Y[0];
  for s:=1 to m-1 do
    begin
      Zn:=C[s-1]+4; C[s]:=-1/Zn;
      D[s]:=(6*y[s]-D[s-1])/Zn;
    end;
  B[m]:=y[m];
  for s:=m-1 downto 0 do
    B[s]:=C[s]*B[s+1]+D[s];
  B[-1]:=2*B[0]-B[1];
```

```
  B[m+1]:=2*B[m]-B[m-1]; s:=s2;
  if s>m-1
    then Bsplintk:=Y[m]
    else Bsplintk:=B[s-1]*(1-x)*sqr(1-x)/6+ B[s]*(2/3+0.5*
         (x-2)*sqr(x))+B[s+1]*(2/3-0.5*(x+1)*
          sqr(x-1))+B[s+2]*x*sqr(x)/6
end;
```

Можна запропонувати також інтерполяцію з допомогою ковзаючого інтерполяційного полінома-підпрограма *IntPolk* .

```
function IntPolk(My:Coefr;k:integer;x:real):real;
type Matr=array[1..15, 1..16] of real;
var m, n, z, s, s1, s2, :integer;
    Y, A: Coef; B:Matr;
begin
  m:=round(My[-1]); s:=trunc(x); x:=frac(x);
  s1:=s; s2:=k;
  if s<k then
    begin
      s1:=k; s2:=s
    end;
  if s>m-k then
    begin
      s1:=m-k; s2:=s-m+2*k
    end;
  for s:=-k to k do Y[s+k]:=My[s+s1]);
  m:=2*k;
  for z:=1 to n+1 do
    begin
      B[z,1]:=1; B[z,n+2]:=Y[z-1];
      for s:=2 to n+1 do
```

```
      B[z,s]:=B[z,s-1]*(z-1)
    end;
  SystUr(n+1,B, A); A[-1]:=n;
  for s:=0 to n do
    A[s]:=A[s+1];
  IntPolk:=HorReal(A, s2+x)
end;
```

Як видно з підпрограми *IntPolk*, для її використання треба мати доступ до процедури *SystUr*.

Оцінити особливості застосування поліноміальної інтерполяції та згладжування функцій, заданих таблично, можна з допомогою програми *DemSpl* текст якої наведено в додатках.

Розділ 3. Чисельне інтегрування та диференціювання

Лекція 11. Гратчасті функції, поліном Лагранжа

Почнемо з того, що існує деяка неперервна функція *f(x)*, де *x* змінюється в діапазоні від *x*=0 до деякого $x=x_{max}$ (коли початкове значення *x* – не нуль, то шляхом заміни змінної завжди легко привести функцію до заданого діапазону).

А тепер приймемо, що в заданому діапазоні *x* змінюється не неперервно, а з деяким кроком, скажімо, *h*=const. Отже, від неперервного *x* ми тим самим переходимо до дискретного *s*, яке може приймати значення 0,1,2,... за умови, що *x*=*sh*.

Значення *y*, що відповідають дискретному набору значень *x*=*sh*, прийнято називати дискретами, а функцію, яка задається сукупністю дискрет, називають гратчастою. Відповідність між неперервною функцією *f(x)* та одержаною з неї гратчастою f_s, показана на рис. 10.

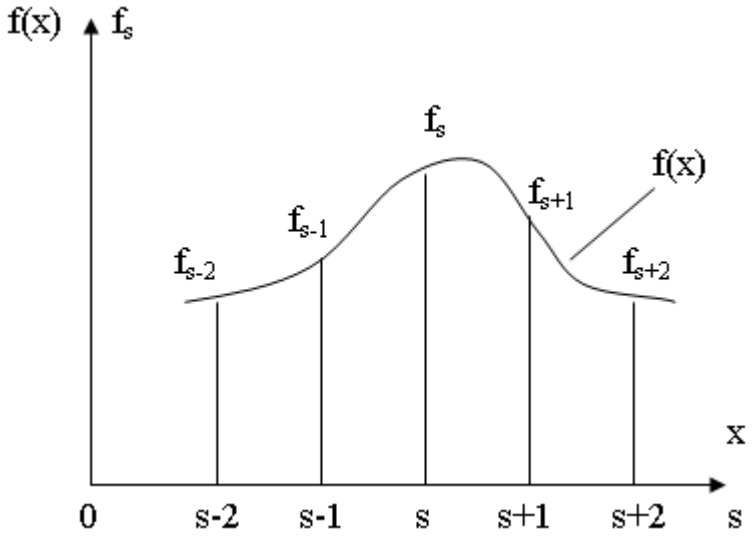

Рис. 10. Неперервна *f(x)* та відповідна їй гратчаста f_s функції

Описаний перехід від неперервної функції *f(x)* до гратчастої f_s є однозначним $f_s = f(x)\big|_{x=sh}$.

Неперервну функцію *f(x)* можна розглядати як огинаючу до гратчастої f_s. За умови, коли структура *f(x)* є невідомою, обернений перехід від гратчастої до неперервної функції стає неоднозначним. Детальніше можливість, доцільність та алгоритми визначення структури та параметрів огинаючих (апроксимуючих) функцій будуть розглянуті нижче.

“Динаміка” неперервної функції характеризується її похідними. Роль похідних для гратчастих функцій виконують так звані різниці.

Перша різниця вперед (перша права різниця) в околі точки з номером *s*

$$\Delta f_s = f_{s+1} - f_s$$

(це аналог першої похідної справа).

Аналогом першої похідної зліва буде перша різниця назад (перша ліва різниця)

$\nabla f_s = f_s - f_{s-1}$.

Перша центральна різниця в точці з номером s

$\delta f_s = f_{s+\frac{1}{2}} - f_{s-\frac{1}{2}}$.

І, нарешті, перша усереднена центральна різниця в точці s

$$\mu\delta f_s = \frac{1}{2}\left(\delta f_{s+\frac{1}{2}} + \delta f_{s-\frac{1}{2}}\right).$$

Якщо в підставити , то одержимо

$$\mu\delta f_s = \frac{f_{s+1} - f_{s-1}}{2}.$$

Аналогічно другим похідним для гратчастої функції визначаються другі різниці.

Друга різниця вперед

$\Delta^2 f_s = \Delta f_{s+1} - \Delta f_s$.

Якщо в підставити , то одержимо

$\Delta^2 f_s = f_{s+2} - 2f_{s+1} + f_s$.

Друга різниця назад

$\nabla^2 f_s = \nabla f_s - \nabla f_{s-1}$

або, з урахуванням ,

$\nabla^2 f_s = f_s - 2f_{s-1} + f_{s-2}$.

Друга центральна різниця

$\delta^2 f_s = \delta f_{s+\frac{1}{2}} - \delta f_{s-\frac{1}{2}}$

або, з урахуванням ,

$\delta^2 f_s = f_{s-1} - 2f_s + f_{s+1}$.

Усереднена центральна різниця для другого, а також для усіх інших парних порядків не визначається – нема потреби, оскільки усереднені різниці вводяться для того, щоб позбутись дробових індексів у формулах, що представляють відповідні центральні різниці через дискрети.

Аналогічно вводяться поняття третіх різниць (аналогів третіх похідних)

$\Delta^3 f_s = \Delta^2 f_{s+1} - \Delta^2 f_s$

або через дискрети

$\Delta^3 f_s = f_{s+3} - 3f_{s+2} + 3f_{s+1} - f_s$.

$\nabla^3 f_s = \nabla^2 f_s - \nabla^2 f_{s-1}$

або

$\nabla^3 f_s = f_s - 3f_{s-1} + 3f_{s-2} - f_{s-3}$.

$\delta^3 f_s = \delta^2 f_{s+\frac{1}{2}} - \delta^2 f_{s-\frac{1}{2}}$

або

$\delta^3 f_s = f_{s+\frac{3}{2}} - 3f_{s+\frac{1}{2}} + 3f_{s-\frac{1}{2}} - f_{s-\frac{3}{2}}$.

$$\mu\delta^3 f_s = \frac{1}{2}\left(\delta^3 f_{s+\frac{1}{2}} + \delta^3 f_{s-\frac{1}{2}}\right)$$

або

$$\mu\delta^3 f_s = \frac{1}{2}\left(f_{s+2} - 2f_{s+1} + 2f_{s-1} - f_{s-2}\right).$$

Цей процес можна продовжувати. В загальному випадку

k-та різниця вперед

$$\Delta^k f_s = \Delta^{k-1} f_{s+1} - \Delta^{k-1} f_s$$

або через дискрети

$$\Delta^k f_s = \sum_{z=0}^{k} (-1)^z \binom{k}{z} f_{s+k-z},$$

тут $\binom{k}{z}$ – біномний коефіцієнт $\binom{k}{z} = \frac{k!}{z!(k-z)!} = C_k^z.$

k-та різниця назад

$$\nabla^k f_s = \nabla^{k-1} f_s - \nabla^{k-1} f_{s-1}$$

або

$$\nabla^k f_s = \sum_{z=0}^{k} (-1)^z \binom{k}{z} f_{s-z}.$$

k-та центральна різниця

$$\delta^k f_s = \delta^{k-1} f_{s+\frac{1}{2}} - \delta^k f_{s-\frac{1}{2}}$$

або

$$\delta^k f_s = \sum_{z=0}^{k} (-1)^z \binom{k}{z} f_{s+\frac{k}{2}-z}.$$

k-та усереднена центральна різниця

$$\mu\delta^k f_s = \frac{1}{2}\left(\delta^k f_{s+\frac{1}{2}} + \delta^k f_{s-\frac{1}{2}}\right)$$

або

$$\mu\delta^k f_s = \frac{1}{2}\left\{ f_{s+\frac{k+1}{2}} + \sum_{z=1}^{k} (-1)^z \left[\binom{k}{z} - \binom{k}{z-1}\right] \cdot f_{s+\frac{k+1}{2}-z} - f_{s-\frac{k+1}{2}} \right\}.$$

Легко бачити, що різниці вперед, назад та центральні пов’язані між собою і їх можна перераховувати одні в інші.

$$\Delta^k f_s = \nabla^k f_{s+k} = \delta^k f_{s+\frac{k}{2}}.$$

$$\nabla^k f_s = \Delta^k f_{s-k} = \delta^k f_{s-\frac{k}{2}}.$$

$$\delta^k f_s = \Delta^k f_{s-\frac{k}{2}} = \nabla^k f_{s+\frac{k}{2}}.$$

Біномний коефіцієнт $\binom{k}{z}$ реалізуємо підпрограмою-функцією *BinomC*.

Представимо формулу для $\binom{k}{z}$ таким чином.

$$\binom{k}{z} = \frac{k!}{z!(k-z)!} = \frac{(k-z)!(k-z+1)(k-z+2)\ldots(k-1)k}{z!(k-z)!}$$

Після скорочення на $(k-z)!$ маємо

$$\binom{k}{z} = \prod_{i=1}^{z} \frac{k-i+1}{i}.$$

```
function BinomC(k,z:integer):integer ;
var i:integer;
   c:real;
begin
 c:=1;
 if z>k
   then c:=0
   else if (z=0) or (z=k)
     then c:=1
     else for i:=1 to z do
          c:=c*(k-i+1)/i;
 BinomC:=round(c)
end;
```

Інтерполяційний поліном Лагранжа. Якщо задано набір вузлів апроксимації x_s, x_{s+1}, x_{s+2},…, x_{s+n} та відповідні їм значення дискрет y_s, y_{s+1}, y_{s+2},…, y_{s+n}, то, як було показано раніше, можна сформувати поліном, значення якого в усіх заданих вузлах співпадатиме з відповідними дискретами.

Це інтерполяційний поліном

$$A(x) = a_0 + a_1 x + a_2 x^2 + \ldots + a_n x^n.$$

Інтерполяційний поліном $A(x)$, сформований таким чином, є єдиним. Але форма його запису не є єдино можливою.

Введемо в розгляд поліном

$$D(x) = (x - x_s)(x - x_{s+1})(x - x_{s+2})\ldots(x - x_{s+n}).$$

Розкладемо на елементарні доданки дріб

$$\frac{A(x)}{D(x)} = \sum_{z=0}^{n} \frac{C_z}{x - x_{s+z}}.$$

Якщо помножити на *D(x)*, отримаємо

$$A(x) = \sum_{z=0}^{n} C_z \prod_{\substack{r=0 \\ r \neq z}}^{n} (x - x_{s+r}).$$

В точці з номером *(s+z)* *x* дорівнюватиме x_{s+z}, а *y* – відповідно y_{s+z}, отже, з випливає, що

$$y_{s+z} = C_z \prod_{\substack{r=0 \\ r \neq z}}^{n} (x_{s+z} - x_{s+r}),$$

звідки

$$C_z = \frac{y_{s+z}}{\prod_{\substack{r=0 \\ r \neq z}}^{n} (x_{s+z} - x_{s+r})}, \quad 0 \le z \le n.$$

Підставляємо в та вводячи замість *A(x)* позначення $L_n(x)$

$$L_n(x) = \sum_{z=0}^{n} y_{s+z} \frac{\prod_{\substack{r=0 \\ r \neq z}}^{n} (x - x_{s+r})}{\prod_{\substack{r=0 \\ r \neq z}}^{n} (x_{s+z} - x_{s+r})}.$$

Поліном $A(x)$ в формі носить назву інтерполяційного полінома Лагранжа. Формула є дуже громіздкою, отже, мало зручною для практичних обчислень, але має велике теоретичне значення.

```
function Lagrange(Mx,My:CoefR; s,n:integer; x:real):real;
var z,r:integer;
P,L:real;
begin
  L:=0;
  for z:=0 to n do
    begin
      P:=1;
      for r:=0 to n do
        if r<>z then
         P:=P*(x-Mx[s+r])/(Mx[s+z]-Mx[s+r]);
        L:=L+My[s+r]*P
    end;
  Lagrange:=L
end;
```

При користуванні підпрограмою *Lagrange* треба слідкувати за тим, щоб

$$Mx[0] \le x \le Mx[m]$$

за умови, що

$$Mx[s+1] \ge Mx[s], \quad 1 \le s \le m-1,$$

де $m := round(Mx[-1])$.

Крім того має виконуватись умова

$$s + n \le m.$$

Лекція 12. Інтерполяційні поліноми Ньютона, Гауса, Стірлінга, Бесселя

Будемо шукати інтерполяційний поліном у формі

$$A(x) = b_0 + b_1(x - x_s) + b_2(x - x_s)(x - x_{s+1}) + b_3(x - x_s)(x - x_{s+1})(x - x_{s+2}) + \ldots + \\ + b_n(x - x_s)(x - x_{s+1})(x - x_{s+2})\ldots(x - x_{s+n-1})$$

В точці $x=x_s$ інтерполяційний поліном має дорівнювати y_s, отже,

$b_0 = y_s$.

в точці $x=x_{s+1}$

$b_0 + b_1 h = y_{s+1}$,

$$b_1 = \frac{y_{s+1} - y_s}{h},$$

або

$$b_1 = \frac{\Delta y_s}{h}.$$

В точці $x=x_{s+2}$

$b_0 + b_1 2h + b_2 2h \cdot h = y_{s+2}$,

звідки

$$b_2 = \frac{y_{s+2} - 2(y_{s+1} - y_s) - y_s}{2h^2} = \frac{y_{s+2} - 2y_{s+1} + y_s}{2h^2}$$

або

$$b_2 = \frac{\Delta^2 y_s}{2!h^2}.$$

Продовжуючи цей процес далі, можна переконатись, що в загальному випадку

$$b_z = \frac{\Delta^z y_s}{z!h^z}, \quad z = 0,1,2,\ldots,n.$$

Позначимо

$$\frac{x - x_s}{h} = u,$$

тобто $x = x_s + uh,$

Тоді

$$\frac{x - x_{s+1}}{h} = \frac{x - (x_s + h)}{h} = u - 1,$$

$$\frac{x - x_{s+2}}{h} = \frac{x - (x_s + 2h)}{h} = u - 2,$$

$$\frac{x - x_{s+n-1}}{h} = u - (n - 1).$$

Якщо замість $A(x)$ ввести позначення $N_n^I(x_s + uh)$, то поліном набуває вигляду

$$N_n^I(x_s+uh)=y_s+u\Delta y_s+\frac{u(u-1)}{2!}\Delta^2 y_s+\frac{u(u-1)(u-2)}{3!}\Delta^3 y_s+\ldots+$$
$$+\frac{u(u-1)(u-2)\ldots[u-(n-1)]}{n!}\Delta^n y_s.$$

Це так званий I інтерполяційний поліном Ньютона або формула Ньютона для інтерполювання вперед.

Інтерполяційний поліном можна шукати і в дещо іншій формі

$$A(x)=C_0+C_1(x-x_s)+C_2(x-x_s)(x-x_{s-1})+C_3(x-x_s)(x-x_{s-1})(x-x_{s-2})+\ldots+$$
$$+C_n(x-x_s)(x-x_{s-1})(x-x_{s-2})\ldots(x-x_{s-(n-1)}).$$

Розглядаючи в точках x_s, x_{s-1}, x_{s-2} і т.д. можна показати, що

$$C_z=\frac{\nabla^z y_s}{z!h^z}=\frac{\Delta^z y_{s-z}}{z!h^z}.$$

Замінюючи x на x_s+u, а $A(x)$ на $N_n^{II}(x_s+uh)$ одержуємо II інтерполяційний поліном Ньютона або формулу Ньютона для інтерполювання назад

$$N_n^{II}(x_s+uh)=y_s+u\Delta y_{s-1}+\frac{u(u+1)}{2!}\Delta^2 y_{s-2}+\frac{u(u+1)(u+2)}{3!}\Delta^3 y_{s-3}+\ldots+$$
$$+\frac{u(u+1)(u+2)\ldots[u+n-1]}{n!}\Delta^n y_{s-n}.$$

Гаусса, Стірлінга та Бесселя

Будемо шукати інтерполяційний поліном $A(x)$ у вигляді

$$A(x)=D_0+D_1(x-x_s)+D_2(x-x_s)(x-x_{s+1})+D_3(x-x_s)(x-x_{s+1})(x-x_{s-1})+$$
$$\ldots+D_n(x-x_s)(x-x_{s+1})(x-x_{s-1})\ldots(x-x_k),$$

де

$$k=\begin{cases}s+\dfrac{n+1}{2}, & \text{якщо } n-\text{непарне},\\ s-\dfrac{n}{2}, & \text{якщо } n-\text{парне}.\end{cases}$$

Розглядаючи при x_s, x_{s+1}, x_{s-1},x_{s+2},... знаходимо, що

$$D_0=y_s,$$

$$D_1=\frac{\delta y_{s+\frac{1}{2}}}{h},$$

$$D_2=\frac{\delta^2 y_s}{2!h^2},$$

$$D_3=\frac{\delta^3 y_{s+\frac{1}{2}}}{3!h^3}.$$

I, в загальному випадку

$$D_z = \begin{cases} \dfrac{\delta^z y_s}{z!h^z}, & \text{якщо } z\text{ - парне,} \\ \dfrac{\delta^z y_{s+\frac{1}{2}}}{z!h^z}, & \text{якщо } z\text{ - непарне.} \end{cases}$$

Переходячи до змінної u та позначаючи $A(x)$ як $G_n^I(x_s+uh)$, одержуємо I інтерполяційний поліном Гауса

$$G_n^I(x_s+uh) = y_s + u\delta\Delta y_{s+\frac{1}{2}} + \frac{u(u-1)}{2!}\delta^2 y_s + \frac{u(u^2-1)}{3!}\delta^3 y_{s+\frac{1}{2}} +$$
$$+ \frac{u(u^2-1)(u-2)}{4!}\delta^4 y_s + \frac{u(u^2-1)(u^2-2^2)}{5!}\delta^5 y_{s+\frac{1}{2}} + \dots$$

Дещо змінимо порядок "обходу" вузлів інтерполяції порівняно з прийнятим у формулі даного параграфу, а саме почнемо з точки x_{s-1}. Таким чином, шукатимемо інтерполяційний поліном у вигляді

$$A(x) = E_0 + E_1(x-x_s) + E_2(x-x_s)(x-x_{s-1}) + E_3(x-x_s)(x-x_{s-1})(x-x_{s+1}) + \dots +$$
$$+ E_n(x-x_s)(x-x_{s-1})(x-x_{s+1})\dots(x-x_k),$$

де

$$k = \begin{cases} s - \dfrac{n+1}{2}, & \text{якщо } n - \text{непарне,} \\ s + \dfrac{n}{2}, & \text{якщо } n - \text{парне.} \end{cases}$$

Розглядаючи при x_s, x_{s-1}, x_{s+1}, x_{s-2},... одержуємо послідовно

$$E_0 = y_s,$$

$$E_1 = \frac{\delta y_{s-\frac{1}{2}}}{h},$$

$$E_2 = \frac{\delta^2 y_s}{2!h^2}.$$

I, в загальному випадку,

$$E_z = \begin{cases} \dfrac{\delta^z y_{s-\frac{1}{2}}}{z!h^z}, & \text{якщо } z\text{ - непарне,} \\ \dfrac{\delta^z y_s}{z!h^z}, & \text{якщо } z\text{ - парне.} \end{cases}$$

Одержаний поліном носить назву II інтерполяційного полінома Гаусса.

$$G_n^{II}(x_s+uh)=y_s+u\delta y_{s-\frac{1}{2}}+\frac{u(u+1)}{2!}\delta^2 y_s+\frac{u(u^2-1)}{3!}\delta^3 y_{s-\frac{1}{2}}+$$
$$+\frac{u(u^2-1)(u+2)}{4!}\delta^4 y_s+\dots$$

Додаючи перший і другий інтерполяційні поліноми Гаусса та ділячи суму навпіл, одержимо інтерполяційний поліном Стірлінга.

$$S_n(x_s+uh)=y_s+u\mu\delta y_s+\frac{u^2}{2!}\delta^2 y_s+\frac{u(u^2-1)}{3!}\mu\delta^3 y_s+\frac{u^2(u^2-1)}{4!}\delta^4 y_s+\dots$$

І нарешті, якщо взяти півсуму II інтерполяційного полінома Гаусса в точці (*s*+1) та I інтерполяційного полінома Гаусса в точці *s*, то одержимо інтерполяційний поліном Бесселя.

$$B_n(x_s+uh)=\frac{y_s+y_{s+1}}{2}+\left(u-\frac{1}{2}\right)\delta y_{s+\frac{1}{2}}+\frac{u(u-1)}{2!}\cdot\frac{\delta^2 y_s+\delta^2 y_{s+1}}{2}+$$
$$+\frac{u(u-1)\left(u-\frac{1}{2}\right)}{3!}\delta^3 y_{s+\frac{1}{2}}+\frac{u(u^2-1)(u-2)}{4!}\cdot\frac{\delta^4 y_s+\delta^4 y_{s+1}}{2}+\dots$$

Формула Бесселя, до речі, зручна для інтерполяції “на середину”, коли $u=\frac{1}{2}$

$$B_n\left(x_s+\frac{h}{2}\right)=\frac{y_s+y_{s+1}}{2}+\frac{u(u-1)}{2!}\cdot\frac{\delta^2 y_s+\delta^2 y_{s+1}}{2}+$$
$$+\frac{u(u^2-1)(u-2)}{4!}\cdot\frac{\delta^4 y_s+\delta^4 y_{s+1}}{2}+\dots$$

Лекція 13. Формули чисельного інтегрування

Нехай досліджувана функція задана звичайним кубічним інтерполяційним сплайном

$\Psi_s(x)=a_{0,s}+a_{1,s}(x-x_{s-1})+a_{2,s}(x-x_{s-1})^2+a_{3,s}(x-x_{s-1})^3$.

Коефіцієнти $a_0, a_{1,s}, a_{2,s}, a_{3,s}, 1\le s\le m$ утворюють масиви *A*0, *A*1, *A*2, *A*3:*CoefR*.

Диференціюємо послідовно двічі вираз

$\Psi_s'(x)=a_{1,s}+2a_{2,s}(x-x_{s-1})+3a_{3,s}(x-x_{s-1})^2$,

$\Psi_s''(x)=2a_{2,s}+6a_{3,s}(x-x_{s-1})$.

Якщо *x* – нормована, тобто відлік ведеться від вузла з номером 0, а крок *h* прийнято за 1, тобто $0\le s\le m$, і під такий *x* сформовано масиви *A*0, *A*1, *A*2, *A*3:*CoefR*, то вважаючи ці масиви глобальними, можна обчислення значень першої *Y*1 та другої *Y*2 похідних оформити у вигляді процедури *DifSpl*.

```
procedure DifSpl(x:real; var Y1,Y2:real);
var m,s:integer;
   h:real;
begin
 m:=round(A0[-1]); h:=A0[m+1];
 s:=trunc(x)+1; x:=frac(x);
 Y1:=(A1[s]+2*A2[s]*x+3*A3[s]*sqr(x))/h;
 Y2:=(2*A2[s]+6*A3[s]*x)/sqr(h)
end;
```

Як видно з процедури *DifSpl*, треба потурбуватись про те, щоб в комірці *A*0[*m*+1] було записано значення кроку *h* між вузлами інтерполяції.

Якщо ж на базі вузлів з номерами $0 \le s \le m$ сформовано глобальний масив *B*:*CoefR* коефіцієнтів кубічних інтерполяційних *B*-сплайнів, то під *x*, що нормований до діапазону 0..*m*, можна записати процедуру *DifBSpl*. Для обчислення першої *Y*1 та другої *Y*2 похідних використовується вираз для самого *Y* (нульової похідної), реалізований в підпрограмі-функції *Bsplint* – див. параграф 3.7. Там ми мали

$$y(x) = B[s-1]\cdot(1-x)^3/6 + B[s]\cdot\left(2/3 + x^2(x-2)\right) + B[s]\cdot\left(2/3 - (x-1)^2(x+1)\right) + \\ + B[s+2]\cdot x^3/6.$$

Першу *Y*1 та другу *Y*2 похідні від обчислюються в підпрограмі *DifBSpl*

```
procedure DifBSpl(x:real; h:real; var Y1,Y2:real);
var s:integer;
begin
 s:=trunc(x); x:=frac(x);
 Y1:=(-B[s-1]*sqr(1-x)/2+B[s]*(3*sqr(x)-4*x)+B[s+1]*
 (-3*sqr(x)+2*x+1)+B[s+2]*sqr(x)/2)/h;
 Y2:=(B[s-1]*(1-x)+B[s]*(6*x-4)+B[s+1]*
 (-6*x+2)+B[s+2]*x)/sqr(h)
end;
```

Тут значення кроку *h* ми змушені передавати в списку параметрів, оскільки в масиві *B*:*CoefR*, де *type CoefR*=*array*[-1..201] *of real*, якщо *m*=200, не знаходиться місця для *h* (масив *B* займає комірки від –1 до (*m*+1) включно)

Нехай підінтегральна функція *y(x)* задана масивами *A*0, *A*1, *A*2, *A*3:*CoefR* коефіцієнтів інтерполяційного сплайна

$$\Psi_s(x) = a_{0,s} + a_{1,s}(x - x_{s-1}) + a_{2,s}(x - x_{s-1})^2 + a_{3,s}(x - x_{s-1})^3, \quad 1 \le s \le m.$$

де $x_0 = a, \quad x_m = b, \quad h = (b-a)/m.$

Тоді

$$\int_a^b y(x)dx \approx \sum_{s=1}^{m}\left[a_{0,s} + a_{1,s}(x - x_{s-1}) + a_{2,s}(x - x_{s-1})^2 + a_{3,s}(x - x_{s-1})^3\right]dx.$$

Вводимо заміну змінної

$$\frac{x-x_{s-1}}{h}=u,$$

тоді

$$\int_a^b y(x)dx \approx h\sum_{s=1}^{m}\int_0^1\left[a_{0,s}+a_{1,s}u+a_{2,s}u^2+a_{3,s}u^3\right]du.$$

Після інтегрування маємо

$$\int_a^b y(x)dx \approx h\sum_{s=1}^{m}\left(a_{0,s}+\frac{a_{1,s}}{2}+\frac{a_{2,s}}{3}+\frac{a_{3,s}}{4}\right).$$

Формулу реалізує підпрограма *IntegrS*.

```
function IntegrS(A0,A1,A2,A3:CoefR):real;
var s,m:integer;
   h,s0,s1,s2,s3:real;
begin
 m:=round(A0[-1]); h:=A0[m+1];
 s0:=0; s1:=0; s2:=0; s3:=0;
 for s:=1 to m do
   begin
     s0:=s0+A0[s];
     s1:=s1+A1[s];
     s2:=s2+A2[s];
     s3:=s3+A3[s]
   end;
 IntegrS:=h*(s0+s1/2+s2/3+s3/4)
end;
```

А тепер розглянемо випадок, коли підінтегральна функція задана масивом *B:CoefR* коефіцієнтів при елементарних інтерполяційних кубічних *B*-сплайнах.

Якщо x відраховується від $x_s, 0\le s\le m, x_0=a, x_m=b, h=(b-a)/m,$ то в межах s-го інтервалу $x_s\le x\le x_{s+1}$ підінтегральна функція, як це випливає з підпрограми *Bsplint*, наведеної в параграфі 3.7,буде

$$\Psi_s(v)=b_{s-1}(1-v)^3/6+b_s\left[2/3+0.5v^2(v-2)\right]+$$
$$+b_{s+1}\left[2/3-0.5(v-1)^2(v+1)\right]+b_{s+2}\,v^3/6,$$

де $v=x-x_s$.

Розкриваємо дужки

$$\Psi_s(v)=b_{s-1}\left(1-3v+3v^2-v^3\right)/6+b_s\left[2/3+\left(v^3-2v^2\right)/2\right]+$$
$$+b_{s+1}\left[2/3-\left(v^3-v^2-v+1\right)/2\right]+b_{s+2}\,v^3/6.$$

Вводимо заміну змінної

$$\frac{x-x_s}{h}=\frac{v}{h}=u.$$

Тоді інтеграл в цілому дорівнюватиме

$$\int_a^b y(x)dx = h\sum_{s=0}^{m-1}\int_0^1 \left\{b_{s-1}\left(1-3u+3u^2-u^3\right)/6 + b_s\left[2/3+\left(u^3-2u^2\right)/2\right]+\right.$$
$$\left. + b_{s+1}\left[2/3-\left(u^3-u^2-u+1\right)/2\right] + b_{s+2}\,u^3/6\right\}du.$$

Беремо інтеграл

$$\int_a^b y(x)dx = h\sum_{s=0}^{m-1}\left\{ b_{s-1}\left(1-\frac{3}{2}+1-\frac{1}{4}\right)\Big/6 + b_s\left[2/3+\left(\frac{1}{4}-\frac{2}{3}\right)\Big/2\right]+\right.$$
$$\left. + b_{s+1}\left[2/3-\left(\frac{1}{4}-\frac{1}{3}-\frac{1}{2}+1\right)\Big/2\right] + b_{s+2}\,\frac{1}{24}\right\}=$$
$$= h\sum_{s=0}^{m-1}\left(b_{s-1}\,\frac{1}{24} + b_s\,\frac{11}{24} + b_{s+1}\,\frac{11}{24} + b_{s+2}\,\frac{1}{24}\right).$$

Остаточно

$$\int_a^b y(x)dx = \frac{h}{24}\sum_{s=0}^{m-1}\left(b_{s-1}+11b_s+11b_{s+1}+b_{s+2}\right).$$

Обчислення інтеграла за формулою реалізує підпрограма *IntegrBS*.

```
function IntegrBS(B:CoefR; m:integer; h:real):real;
var s:integer; Sum:real;
begin
  Sum:=0;
  for s:=0 to m-1 do
    Sum:=Sum+B[s-1]+11*B[s]+11*B[s+1]+B[s+2];
  IntegrBS:=h*Sum/24
end;
```

Лекція 14. Принцип Рунге

Правило Рунге - правило оцінки похибки чисельних методів, було запропоновано К. Рунге на початку 20 століття.

Основна ідея полягає в обчисленні наближення обраним методом з кроком h, а потім з кроком h / 2, і подальшому розгляді різниць похибок для цих двох обчислень.

Оцінка точності обчислення певного інтеграла

Інтеграл обчислюється за обраною формулою (прямокутників, трапецій, парабол Сімпсона) при числі кроків, рівному n, а потім при числі кроків, рівному 2n. Похибка обчислення значення інтеграла при числі кроків, рівному 2n, визначається за формулою Рунге:

$\Delta_{2n} \approx \Theta |I_{2n} - I_n|$, для формул прямокутників і трапецій $\Theta = 1/3$,, а для формули Сімпсона $\Theta = 1/15$.

Таким чином, інтеграл обчислюється для послідовних значень числа кроків $N = n_0, 2n_0, 4n_0, \ldots$, де n0 - початкова число кроків. Процес обчислень

закінчується, коли для чергового значення N буде виконана умова $\Delta_{2n} < \epsilon$, де ε - задана точність.

Оцінка точності чисельного рішення рівнянь

Також застосовується для оцінки точності рішень звичайних диференціальних рівнянь на регулярних сітках. Для оцінки потрібно вирішити задачу на 2 сітках, один раз з кроком h (yi, h) і другий - з кроком h / 2 (yi, h / 2). Формула

| yi, h-yi, h / 2 | 2p-1 дає похибку рішення yi, h / 2. Під p розуміється порядок точності використаного чисельного методу. Наприклад, для чисельного методу, що має четвертий порядок точності, формула приймає вигляд:

| yi, h-yi, h / 2 |

Існує сімейство методів Рунге-Кутта числового інтегрування звичайних диференціальних рівнянь. Нам здається, що краще говорити про варіанти реалізації одного методу, суть якого розглянемо для початку на прикладі окремо взятого диференціального рівняння в канонічній формі .

Черговий крок числового інтегрування диференціального рівняння від $t_\nu = \nu\tau$ до $t_{\nu+1} = (\nu+1)\tau$ організуємо таким чином: спочатку обчислюємо q допоміжних функцій R_s $(1 \le s \le q)$, після чого виконуємо робочий крок:

$$\begin{cases} R_1 = \tau f(y_\nu, t_\nu); \\ \ldots\ldots \\ R_q = \tau f(y^{(q)}, t^{(q)}); \\ y_{\nu+1} = y_\nu + \sum\limits_{s=1}^{q} p_s R_s, \end{cases}$$

де

$$y^{(s)} = y_\nu + \sum_{i=1}^{q} \beta_{s,i} R_i, \; 2 \le s \le q,$$

$$t^{(s)} = t_\nu + \alpha_s \tau.$$

Розглянемо тепер питання вибору коефіцієнтів $\alpha_s (1 \le s \le q)$, $\beta_{s,i} (2 \le s \le q, 1 \le i \le s-1)$, $p_s (1 \le s \le q)$.

Якщо позначити точний розв'язок на кроці τ як $y(t+\tau)$, то похибка на кроці τ буде

$$\varepsilon(\tau) = y(t+\tau)\,\big|_{t=t_\nu} - y_{\nu+1}.$$

Якщо нам вдасться підібрати коефіцієнти α_s , β_{is} , p_s таким чином, що для початку кроку ($\tau = 0$)

$$\varepsilon(0) = \varepsilon(0) = ... = \varepsilon^{(p)}(0) = 0,$$

то розкладаючи функцію в ряд Тейлора в околі точки t_ν

$$\varepsilon(\tau) = \sum_{z=0}^{\rho} \frac{\varepsilon^{(z)}(0)}{z!}\tau^z + \frac{\varepsilon^{(\rho+1)}(\tau)}{(\rho+1)!}\tau^{\rho+1}$$

і, враховуючи, , одержуємо

$$\varepsilon(\tau) = \frac{\varepsilon^{(\rho+1)}(\theta\tau)}{(\rho+1)!}\tau^{\rho+1},$$

де $0 \le \theta \le 1$.

Величина $\varepsilon(\tau)$ в формулі називається похибкою методу на кроці, а ρ - порядком похибки методу.

При q=1 в формулах матимемо:

$$\begin{cases} R_1 = \tau f(y_\nu, t_\nu); \\ y_{\nu+1} = y_\nu + p_1 \tau f(y_\nu, t_\nu). \end{cases}$$

Похибка на кроці буде

$$\varepsilon(t) = y(t_1 + \tau) - y_\nu - p_1 \tau f(y_\nu, t_\nu).$$

Диференціюємо за τ

$$\varepsilon'(t) = y'(t_1 + \tau) - p_1 f(y_\nu, t_\nu).$$

Диференціюємо тепер за τ

$$\varepsilon''(t) = y''(t_1 + \tau).$$

При $\tau = 0$, та дають:

$$\varepsilon(0) = 0,$$

адже $y(t_1 + \tau) = y_\nu$.

$$\varepsilon'(0) = (1 - p_1) f(y_\nu, t_\nu),$$

адже $y'(t_1 + 0) = f(y_\nu, t_\nu)$.

$$\varepsilon''(0) = y''(t_1).$$

Як слідує з при $p_1 = 1$, $\varepsilon'(0) = 0$, отже формули набувають вигляду:

$$\begin{cases} R_1 = \tau f(y_\nu, t_\nu); \\ y_{\nu+1} = y_\nu + R_1. \end{cases}, \quad \rho = 1.$$

Але ж це відомий нам метод Ейлера. Оскільки $\varepsilon''(0) \neq 0$, то в загальному випадку метод Ейлера має порядок точності $\rho = 1$.

А тепер розглянемо варіант, коли q=2.

Система співвідношень набуває вигляду:

$$\begin{cases} R_1 = \tau f(y_\nu, t_\nu); \\ R_2 = \tau f(y^{(2)}, t^{(2)}); \\ y_{\nu+1} = y_\nu + p_1 R_1 + p_2 R_2, \end{cases}$$

де $y^{(2)} = y_\nu + \beta_{2,1} R_1$,

$t^{(2)} = t_\nu + a_{2,1}\tau$.

Похибка на кроці τ згідно буде

$$\varepsilon(t) = y(t_1 + \tau) - y_\nu - p_1 \tau f(y_\nu, t_\nu) - p_2 \tau f(y^{(2)}, t^{(2)}).$$

Диференціюємо за τ

$$\varepsilon'(t) = y'(t_1 + \tau) - y_\nu - p_1 f(y_\nu, t_\nu) - p_2 f(y^{(2)}, t^{(2)}) -$$
$$p_2 \tau(\beta_{2,1} f(y_\nu, t_\nu) f_y(y^{(2)}, t^{(2)}) + \alpha_2 f_t(y^{(2)}, t^{(2)})),$$

де $f_t = \dfrac{df}{dt}$,

$f_y = \dfrac{df}{dy}$.

А тепер диференціюємо :

$$\varepsilon''(t) = y''(t_\nu + \tau) - p_2(\beta_{2,1} f(y_\nu, t_\nu) f_y(y^{(2)}, t^{(2)}) +$$
$$+ \alpha_2 f_t(y^{(2)}, t^{(2)})) - p_2(\beta_{2,1} f(y_\nu, t_\nu) f_y(y^{(2)}, t^{(2)}) +$$
$$+ \alpha_2 f_t(y^{(2)}, t^{(2)})) - p_2 \tau(\beta_{2,1}^2 \ f(y_\nu, t_\nu))^2 f_y(y^{(2)}, t^{(2)}) +$$
$$+ \beta_{2,1} f(y_\nu, t_\nu) \alpha_2 f_{yt}(y^{(2)}, t^{(2)})) + \beta_{2,1}^2 \ f_{yy}(y_\nu, t_\nu)(f(y_\nu, t_\nu))^2 +$$
$$+ \alpha_2 \beta_{2,1} f(y_\nu, t_\nu) f_{ty}(y^{(2)}, t^{(2)}) + \alpha_2^2 f_u(y^{(2)}, t^{(2)})$$

або, після зведення подібних,:

$$\varepsilon''(t) = y''(t_\nu + \tau) - 2p_2(\beta_{2,1} f(y_\nu, t_\nu) f_y(y^{(2)}, t^{(2)}) +$$
$$+ \alpha_2 f_t(y^{(2)}, t^{(2)})) - p_2 \tau(2\alpha_2 \beta_{2,1} f(y_\nu, t_\nu) +$$
$$f_{yt}(y^{(2)}, t^{(2)}) + \beta_{2,1} f_{yy}(y^{(2)}, t^{(2)})((f \ (y_\nu, t_\nu))^2 +$$
$$+ \alpha_2^2 f_u(y^{(2)}, t^{(2)})).$$

Диференціюванням отримуємо

$$\varepsilon'''(t) = y'''(t_\nu + \tau) - 3p_2(\alpha_2^2 f_u(y^{(2)}, t^{(2)}) +$$
$$+ 2\alpha_2^2 \beta_{2,1} f_u(y^{(2)}, t^{(2)}) f(y_\nu, t_\nu) +$$
$$+ \beta_{2,1}^2 f_{yy}(y^{(2)}, t^{(2)})(f(y_\nu, t_\nu))^2).$$

Розглядаємо тепер , , , при $\tau = 0$

$$\varepsilon(0) = y_\nu - y_\nu = 0;$$

$$\varepsilon'(0) = f(y_\nu, t_\nu) - p_1 f(y_\nu, t_\nu) - p_2 f(y_\nu, t_\nu) = (1 - p_1 - p_2) f(y_\nu, t_\nu);$$

$$\varepsilon''(0) = f_1(y_\nu, t_\nu) - f(y_\nu, t_\nu) f_y(y_\nu, t_\nu) - 2p_2(\beta_{2,1} f(y_\nu, t_\nu) f_y(y_\nu, t_\nu) +$$
$$+ \alpha_2 f_t(y_\nu, t_\nu) = (1 - 2p_2 a_2) f(y_\nu, t_\nu) +$$
$$+ (1 - p_2 \beta_{2,1}) f_y(y_\nu, t_\nu) f(y_\nu, t_\nu);$$

$$\varepsilon'''(0) = (1 - 3p_2 a_2^2) f_u(y_\nu, t_\nu) + (2 - 6p_2 \alpha_2 \beta_{2,1}) f_{ty}(y_\nu, t_\nu) f(y_\nu, t_\nu) +$$

$$+(1-3p_2\beta_{2,1}^2)f_{yy}(y_\nu,t_\nu)(f(y_\nu,t_\nu))^2+f(y_\nu,t_\nu)y''(t_\nu).$$

Співвідношення $\varepsilon(0)=0$ виконується автоматично.

Перша похідна $\varepsilon'(0)$ стає рівною 0 за умови

$$(1-p_1-p_2)=0.$$

Друга похідна $\varepsilon''(0)$ буде нулем за виконання двох умов:

$$1-2p_2\alpha_2=0,$$

та

$$1-2p_2\beta_{2,1}=0.$$

Отже, маємо систему трьох рівнянь , , відносно чотирьох невідомих $\alpha_2,\beta_{2,1}$, p_1 та p_2. Ця система має безліч розв'язків. Наприклад, такий

$$\begin{cases} p_1=p_2=\dfrac{1}{2}; \\ \alpha_2=1; \\ \beta_{2,1}=1. \end{cases}$$

Розрахункова схема при цьому набуває вигляду:

$$\begin{cases} R_1=\tau f(y_\nu,t_\nu); \\ R_2=\tau f(y_\nu+R_1,t_\nu+\tau); \qquad \rho=2 \\ y_{\nu+1}=y_\nu+\dfrac{R_1+R_2}{2}. \end{cases}$$

Це, як легко бачити, метод Ейлера-Коші. Систему - можна задовольнити також при

$$\begin{cases} p_1=0; \\ p_2=1; \\ \alpha_2=\dfrac{1}{2}; \\ \beta_{2,1}=\dfrac{1}{2}. \end{cases}$$

Розрахункова схема при цьому буде

$$\begin{cases} R_1=\tau\cdot f(y_\nu,t_\nu); \\ R_2=\tau\cdot f(y_\nu+\dfrac{R_1}{2},t_\nu+\dfrac{\tau}{2}); \ \rho=2 \\ y_{\nu+1}=y_\nu+R_2. \end{cases}$$

Це перший модифікований метод Ейлера. Свободою вибору частини невідомих в розв'язку системи - можна скористатись, щоб мінімізувати значення $\varepsilon'''(0)$. Так, наприклад, при

$$\begin{cases} p_1 = \dfrac{1}{4}; \\ p_2 = \dfrac{3}{4}; \\ \alpha_2 = \beta_{2,1} = \dfrac{2}{3} \end{cases}$$

третя похідна $\varepsilon'''(0) = f_y(y_\nu, t_\nu) y''(t_\nu)$.

Цій комбінації коефіцієнтів відповідає схема:

$$\begin{cases} R_1 = \tau \cdot f(y_\nu, t_\nu); \\ R_2 = \tau \cdot f(y_\nu + \dfrac{2}{3} R_1, t_\nu + \dfrac{2}{3}\tau); \qquad \rho = 2 \\ y_{\nu+1} = y_\nu + \dfrac{R_1 + 3R_2}{4}. \end{cases}$$

Формули відносяться до групи методів Рунге-Кутта 2-го порядка точності (той же порядок точності мають вищезгадані метод Ейлера-Коші та перший модифікований метод Ейлера).

При q=3, s=3 необхідні умови щодо коефіцієнтів виглядають так:

$$\begin{cases} \alpha_2 = \beta_{2,1}; \\ \alpha_3 = \beta_{3,1} + \beta_{3,2}; \\ \alpha_3(\alpha_3 - \alpha_2) - \beta_{3,2}\alpha_2(2 - \alpha_2) = 0; \\ p_3 \beta_{3,2} \alpha_2 = \dfrac{1}{6}; \\ p_2 \alpha_2 + p_3 \alpha_3 = \dfrac{1}{2}; \\ p_1 + p_2 + p_3 = 1. \end{cases}$$

Це система 6 рівнянь з 8 невідомими. Зрозуміло, що вона має безліч розв'язків. Часто використовується такий варіант:

$$\begin{cases} R_1 = h \cdot f(y_\nu, t_\nu), \\ R_2 = h \cdot f(y_\nu + \dfrac{R_1}{2}, t_\nu + \dfrac{\tau}{2}), \\ R_3 = h \cdot f(y_\nu - R_1 + 2R_2, t_\nu + \tau), \\ y_{\nu+1} = y_\nu + \dfrac{R_1 + 4R_2 + R_3}{6}; \end{cases} \qquad \rho = 3$$

Якщо *f(y,t)* не залежить від *y*, тобто коли $f_y = 0$, схема перетворюється на відому формулу Симпсона

$$y_{\nu+1} - y_\nu = \frac{\tau}{6}\left[f(t_\nu) + 4 f(t_\nu + \frac{\tau}{2}) + f(t_\nu + \tau) \right].$$

Використовують і такий варіант:

$$\begin{cases} R_1 = h \cdot f(y_\nu, t_\nu); \\ R_2 = h \cdot f(y_\nu + \frac{R_1}{3}, t_\nu + \frac{\tau}{3}); \\ R_3 = h \cdot f(y_\nu + \frac{2R_2}{3}, t_\nu + \frac{2\tau}{3}); \qquad \rho = 3. \\ y_{\nu+1} = y_\nu + \frac{R_1 + 3R_3}{4} \end{cases}$$

або такий

$$\begin{cases} R_1 = h \cdot f(y_\nu, t_\nu); \\ R_2 = h \cdot f(y_\nu + \frac{R_1}{2}, t_\nu + \frac{\tau}{2}); \\ R_3 = h \cdot f(y_\nu + \frac{3R_2}{4}, t_\nu + \frac{3\tau}{4}); \qquad \rho = 3. \\ y_{\nu+1} = y_\nu + \frac{2R_1 + 3R_2 + 4R_3}{9}. \end{cases}$$

При q=4 та s=4 маємо множину розрахункових формул. В літературі наводяться такі формули:

$$\begin{cases} R_1 = \tau \cdot f(y_\nu, t_\nu); \\ R_2 = \tau \cdot f(y_\nu + \frac{R_1}{3}, t_\nu + \frac{\tau}{3}); \\ R_3 = \tau \cdot f(y_\nu - \frac{R_1}{3} + R_2, t_\nu + \frac{2\tau}{3}); \qquad \rho = 4. \\ R_4 = \tau \cdot f(y_\nu + R_1 - R_2 + R_3, t_\nu + \tau); \\ y_{\nu+1} = y_\nu + \frac{R_1 + 3(R_2 + R_3) + R_4}{8}. \end{cases}$$

$$\begin{cases} R_1 = \tau \cdot f(y_\nu, t_\nu); \\ R_2 = \tau \cdot f(y_\nu + \frac{R_1}{4}, t_\nu + \frac{\tau}{4}); \\ R_3 = \tau \cdot f(y_\nu + \frac{R_1}{2} + R_2, t_\nu + \frac{\tau}{2}); \qquad \rho = 4. \\ R_4 = \tau \cdot f(y_\nu + R_1 - 2R_2 + R_3, t_\nu + \tau); \\ y_{\nu+1} = y_\nu + \frac{R_1 + 4R_3 + R_4}{6}. \end{cases}$$

Частіше ж усього серед варіантів методу Рунге-Кутта четвертого порядку рекомендують такий алгоритм

$$\begin{cases} R_1 = \tau \cdot f(y_\nu, t_\nu); \\ R_2 = \tau \cdot f(y_\nu + \frac{R_1}{2}, t_\nu + \frac{\tau}{2}); \\ R_3 = \tau \cdot f(y_\nu + \frac{R_2}{2}, t_\nu + \frac{\tau}{2}); \\ R_4 = \tau \cdot f(y_\nu + R_3, t_\nu + \tau); \\ y_{\nu+1} = y_\nu + \frac{R_1 + 2(R_2 + R_3) + R_4}{6}. \end{cases} \qquad \rho = 4 \ .$$

Алгоритм вважається класичним методом Рунге-Кутта. В багатьох підручниках цей метод часто називають просто "метод Рунге Кутта" (не уточнюючи, що мова йде про метод 4-го порядку точності) або ж "метод Рунге-Кутта четвертого порядку точності".

Прикладом варіанту метода Рунге-Кутта 4-го порядку точності можна розгдядати такий алгоритм:

$$\begin{cases} R_1 = \tau \cdot f(y_\nu, t_\nu); \\ R_2 = \tau \cdot f\left(y_\nu + \frac{R_1}{3}, t_\nu + \frac{\tau}{3}\right); \\ R_3 = \tau \cdot f\left(y_\nu + \frac{4R_1 + 6R_2}{25}, t_\nu + \frac{2}{5}\tau\right); \\ R_4 = \tau \cdot f\left(y_\nu + \frac{R_1 - 12R_2 + 15R_3}{4}, t_\nu + \tau\right); \\ R_5 = \tau \cdot f\left(y_\nu + \frac{6R_1 + 90R_2 - 50R_3 + 8R_4}{81}, t_\nu + \frac{2}{3}\tau\right); \\ R_6 = \tau \cdot f\left(y_\nu + \frac{6R_1 + 36R_2 + 10R_3 + 8R_4}{75}, t_\nu + \frac{4}{5}\tau\right); \\ y_{\nu+1} = y_\nu + \frac{23R_1 + 125R_3 - 81R_5 + 125R_6}{192}; \rho = 5. \end{cases}$$

Реалізуємо класичний метод Рунге-Кутта стосовно до системи канонічних рівнянь :

$$\begin{cases} R_{z,1} = \tau\left(\sum\limits_{s=1}^{n} A_{z,s} y_{s,v} + \sum\limits_{s=1}^{m} B_{z,s} x_{s,v} \right), & 1 \le z \le n; \\ R_{z,2} = \tau\left(\sum\limits_{s=1}^{n} A_{z,s} (y_{s,v} + \frac{1}{2} R_{s,1}) + \sum\limits_{s=1}^{m} B_{z,s} x_{s,v+\frac{1}{2}} \right), & 1 \le z \le n; \\ R_{z,3} = \tau\left(\sum\limits_{s=1}^{n} A_{z,s} (y_{s,v} + \frac{1}{2} R_{s,2}) + \sum\limits_{s=1}^{m} B_{z,s} x_{s,v+\frac{1}{2}} \right), & 1 \le z \le n; \\ R_{z,4} = \tau\left(\sum\limits_{s=1}^{n} A_{z,s} (y_{s,v} + R_{s,3}) + \sum\limits_{s=1}^{m} B_{z,s}, x_{s,v+1} \right), & 1 \le z \le n; \\ y_{z,v+1} = y_{z,v} + \dfrac{R_{z,1} + 2(R_{z,2} + R_{z,3}) + R_{z,4}}{6}, & 1 \le z \le n. \end{cases}$$

Ці формули можна переписати так:

$$\begin{cases} R_{z,1} = \tau\left(C_z + \sum\limits_{s=1}^{m} B_{z,s} x_{s,v} \right), & 1 \le z \le n; \\ R_{z,2} = \tau\left(C_z + \frac{1}{2} \sum\limits_{s=1}^{n} A_{z,s} R_{s,1} + \sum\limits_{s=1}^{m} B_{z,s} x_{s,v+\frac{1}{2}} \right), & 1 \le z \le n; \\ R_{z,3} = \tau\left(C_z + \frac{1}{2} \sum\limits_{s=1}^{n} A_{z,s} R_{s,2} + \sum\limits_{s=1}^{m} B_{z,s} x_{s,v+\frac{1}{2}} \right), & 1 \le z \le n; \\ R_{z,4} = \tau\left(C_z + \sum\limits_{s=1}^{n} A_{z,s} R_{s,3} + \sum\limits_{s=1}^{m} B_{z,s}, x_{s,v+1} \right), & 1 \le z \le n; \\ y_{z,v+1} = y_{z,v} + \dfrac{R_{z,1} + 2(R_{z,2} + R_{z,3}) + R_{z,4}}{6}, & 1 \le z \le n, \end{cases}$$

де

$$C_z = \sum_{s=1}^{n} A_{z,s} y_{s,v} .$$

Алгоритм реалізується в підпрограмі *RunKutSu*.

```
procedure RunKutSu(A,B:Coef2;Mx0,Mxs,Mx1:Coef;Tau:real;
                   var Y:Coef);
var z,s,n,m,w: integer;
    R: array[1..30,1..4] of real;
    Mx,C: Coef;
    Sum,Alreal;
begin
```

```
  n:=round(A[1,-1]); m:=round(B[1,-1]);
  for z:=1 to n do
    begin
      C[z]:=0;
      for s :=1 to n do C[z]:=C[z]+A[z,s]*Y[s];
    end;
  for w := 1 to 4 do
    begin
      case w of
        1:Mx:=Mx0;
      2,3:begin
            Al:=1/2;Mx:=MxS
          end;
        4:begin
            Al:=1;Mx:=Mx1
          end
      end;
      for z:=1 to n do
        begin
          Sb:=0;
          for s:=1 to m do
          Sb:=Sb+B[z,s]*Mx[s];
          if w=1
            then
              R[z,w]:=Tau*(C[z]+Sum)
            else
              begin
                Sa:=0;
                for s:=1 to n do
                  Sa:=Sa+A[z,s]*R[s,w-1]
              end;
          R[z,w]:=Tau*(C[z]+Sb+Sa*Al)
        end
     end;
  for z:=1 to n do
    Y[z]:=Y[z]+(R[z,1]+2*(R[z,2]+R[z,3])+R[z,4])/6
end;
```

Для системи заміщення - алгоритм набуває вигляду:

$$
\begin{cases}
U_{n+1,v} = -\dfrac{1}{a_n}\sum\limits_{s=0}^{n-1} a_s U_{s+1,v}; \\
R_{z,1} = \tau\left(k_z x_v + U_{z+1,v}\right), & 1 \le z \le n; \\
R_{z,2} = \tau\left(k_z x_{v+\frac{1}{2}} + U_{z+1,v} + \dfrac{R_{z,1}}{2}\right), & 1 \le z \le n; \\
R_{z,3} = \tau\left(k_z x_{v+\frac{1}{2}} + U_{z+1,v} + \dfrac{R_{z,2}}{2}\right), & 1 \le z \le n; \\
R_{z,4} = \tau\left(k_z x_{v+1} + U_{z+1,v} + R_{z,3}\right), & 1 \le z \le n; \\
U_{z,v+1} = U_{z,v} + \dfrac{R_{z,1} + 2(R_{z,2} + R_{z,3}) + R_{z,4}}{6}, & 1 \le z \le n; \\
U_{0,v+1} = k_0 x_{v+1} + U_{1,v+1}.
\end{cases}
$$

Алгоритм реалізує підпрограма *RunKutWp*.

```
Procedure RunKutWp(A,K:Coef;x0,xs,x1,Tau:real;
                                              var U:Coef);
var z,s,n:integer;
    t,x,Sum,R1,R2,R3,R4:real;
begin
  n:=round(A[-1]);Sum:=0;
  for s:=0 to n-1 do
    Sum:=Sum+A[s]*U[s+1];
    U[n+1] := -Sum/A[n];
    for z:=1 to n do
      begin
        R1:=Tau*(K[z]*x0+U[z+1]);
        R2:=Tau*(K[z]*xs+U[z+1]+R1/2);
        R3:=Tau*(K[z]*xs+U[z+1]+R2/2);
        R4:=Tau*(K[z]*x1+U[z+1]+R3);
        U[z]:=U[z]+(R1+2*(R2+R3)+R4)/6
      end;
  U[0]:=K[0]*x1+U[1]
end;
```

Лекція 15. Чисельне диференціювання

Диференціювання. Ідея числового диференціювання зводиться до того, що досліджувана функція замінюється інтерполяційним поліномом (в тій чи

іншій формі), після чого похідні від цього полінома приймаються наближено рівними похідним від досліджуваної функції при відповідних значеннях *x* (як у вузлах, так і між ними).

Коли ми маємо інтерполяційний поліном у вигляді
$A(x) = a_0 + a_1 x + a_2 x^2 + \ldots + a_n x^n$,
то диференціювання його не складає проблеми.
Однократне диференціювання полінома реалізує підпрограми *DifPol*.

```
procedure DifPol(A:Coef;var B:Coef);
var n,s:integer;
begin
  n:=round(A[-1]);
  B[-1]:=n-1;
  for s:=0 to n-1 do
    B[s]:=A[s+1]*(s+1)
end;
```

Обчислення значення полінома *B(p)* можна реалізувати в підпрограмі-функції *HorReal - y:=HorReal(B,x).*
Інтерполяційний полінома Лагранжа надто громіздкий і ми його використання для диференціювання розглядати не будемо.
Якщо скористатись І інтерполяційним поліномом Ньютона,

$$y(x) = N_n^I(x_s + uh) = y_s + u\Delta y_s + \frac{u(u-1)}{2!}\Delta^2 y_s + \\ + \frac{u(u-1)(u-2)}{3!}\Delta^3 y_s + \ldots + \frac{u(u-1)(u-2)\ldots[u-(n-1)]}{n!}\Delta^n y_s,$$

то

$$y'(x) = \frac{1}{h}\left[\Delta y_s + \frac{2u-1}{2!}\Delta^2 y_s + \frac{3u^2-6u+2}{3!}\Delta^3 y_s + \ldots\right],$$

$$y''(x) = \frac{1}{h^2}\left[\Delta^2 y_s + \frac{6u-6}{3!}\Delta^3 y_s + \frac{12u^2-36u+22}{4!}\Delta^4 y_s + \ldots\right],$$

$$y'''(x) = \frac{1}{h^3}\left[\Delta^3 y_s + \frac{24u-36}{4!}\Delta^4 y_s + \frac{60u^2-240u+210}{5!}\Delta^5 y_s + \ldots\right],$$

$$y^{IV}(x) = \frac{1}{h^4}\left[\Delta^4 y_s + \frac{120u-240}{5!}\Delta^5 y_s + \frac{360u^2-1800u+2040}{6!}\Delta^6 y_s + \ldots\right],$$

$$y^{V}(x) = \frac{1}{h^5}\left[\Delta^5 y_s + \frac{720u-1800}{6!}\Delta^6 y_s + \ldots\right],$$

Якщо в цих формулах покласти *u*=0, тобто шукати значення похідних у вузлах інтерполяції, то одержимо

$$y'_s = \frac{1}{h}\left[\Delta y_s - \frac{\Delta^2 y_s}{2} + \frac{\Delta^3 y_s}{3} - \frac{\Delta^4 y_s}{4} + \dots\right],$$

$$y''_s = \frac{1}{h^2}\left[\Delta^2 y_s - \Delta^3 y_s + \frac{11}{12}\Delta^4 y_s - \dots\right],$$

$$y'''_s = \frac{1}{h^3}\left[\Delta^3 y_s - \frac{3}{2}\Delta^4 y_s + \frac{7}{4}\Delta^5 y_s - \dots\right],$$

$$y^{IV}{}_s = \frac{1}{h^4}\left[\Delta^4 y_s - 2\Delta^5 y_s + \frac{17}{6}\Delta^6 y_s - \dots\right].$$

Рекомендуємо читачеві порівняти формули - з формулами , , параграфа , щоб упевнитись в їх повній ідентичності.

Якщо скористатись тепер II інтерполяційним поліномом Ньютона

$$y(x) = N_n^{II}(x_s + uh) = y_s + u\nabla y_s + \frac{u(u+1)}{2!}\nabla^2 y_s + \frac{u(u+1)(u+2)}{3!}\nabla^3 y_s + \dots +$$
$$+ \frac{u(u+1)(u+2)\dots[u+n-1]}{n!}\nabla^n y_s.$$

то одержимо відповідно

$$y'(x) = \frac{1}{h}\left[\nabla y_s + \frac{2u+1}{2!}\nabla^2 y_s + \frac{3u^2+6u+2}{3!}\nabla^3 y_s + \right.$$
$$\left. + \frac{4u^3+18u^2+22u+6}{4!}\nabla^4 y_s + \dots\right],$$

$$y''(x) = \frac{1}{h^2}\left[\nabla^2 y_s + \frac{6u+6}{3!}\nabla^3 y_s + \frac{12u^2+36u+22}{4!}\nabla^4 y_s + \dots\right],$$

$$y'''(x) = \frac{1}{h^3}\left[\nabla^3 y_s + \frac{24u+36}{4!}\nabla^4 y_s + \frac{60u^2+240u+210}{5!}\nabla^5 y_s + \dots\right],$$

У вузлах інтерполяції (при u=0)

$$y'(x) = \frac{1}{h}\left[\nabla y_s + \frac{\nabla^2 y_s}{2} + \frac{\nabla^3 y_s}{3} + \frac{\nabla^4 y_s}{4} + \dots\right],$$

$$y''(x) = \frac{1}{h^2}\left[\nabla^2 y_s + \nabla^3 y_s + \frac{11}{12}\nabla^4 y_s + \dots\right],$$

$$y'''(x) = \frac{1}{h^3}\left[\nabla^3 y_s + \frac{3}{2}\nabla^4 y_s + \frac{7}{4}\nabla^5 y_s + \dots\right],$$

Формули - варто порівняти з формулами , , параграфа 4.3.

Візьмемо тепер I інтерполяційний поліном Гаусса

$$y(x)=G_n^I(x_s+uh)=y_s+u\delta\Delta y_{s+\frac{1}{2}}+\frac{u(u-1)}{2!}\delta^2 y_s+\frac{u(u^2-1)}{3!}\delta^3 y_{s+\frac{1}{2}}+$$
$$+\frac{u(u^2-1)(u-2)}{4!}\delta^4 y_s+\frac{u(u^2-1)(u^2-2^2)}{5!}\delta^5 y_{s+\frac{1}{2}}+\dots$$

Звідси

$$y'(x)=\frac{1}{h}\left[\delta y_{s+\frac{1}{2}}+\frac{2u-1}{2!}\delta^2 y_s+\frac{3u^2-1}{3!}\delta^3 y_{s+\frac{1}{2}}+\right.$$
$$\left.+\frac{4u^3-6u^2-2u+2}{4!}\delta^4 y_s+\frac{5u^4-15u^2+4}{5!}\delta^5 y_{s+\frac{1}{2}}\dots\right],$$

$$y''(x)=\frac{1}{h^2}\left[\delta^2 y_s+\frac{6u}{3!}\delta^3 y_{s+\frac{1}{2}}+\frac{12u^2-12u-2}{4!}\delta^4 y_s+\frac{20u^3-30u}{5!}\delta^5 y_{s+\frac{1}{2}}+\dots\right],$$

$$y'''(x)=\frac{1}{h^3}\left[\delta^3 y_{s+\frac{1}{2}}+\frac{244u-12}{4!}\delta^4 y_s+\frac{60u^2-30}{5!}\delta^5 y_{s+\frac{1}{2}}+\dots\right],$$

$$y^{IV}(x)=\frac{1}{h^4}\left[\delta^4 y_s+\frac{120u}{5!}\delta^5 y_{s+\frac{1}{2}}+\frac{360u^2-360u-120}{6!}\delta^6 y_s+\dots\right].$$

При u=0 маємо

$$y'_s=\frac{1}{h}\left[\delta y_{s+\frac{1}{2}}-\frac{\delta^2 y_s}{2}-\frac{\delta^3 y_{s+\frac{1}{2}}}{6}+\frac{\delta^4 y_s}{12}+\frac{\delta^5 y_{s+\frac{1}{2}}}{30}-\dots\right],$$

$$y''_s=\frac{1}{h^2}\left[\delta^2 y_s-\frac{\delta^4 y_s}{12}+\frac{\delta^6 y_s}{90}-\dots\right],$$

$$y'''_s=\frac{1}{h^3}\left[\delta^3 y_{s+\frac{1}{2}}-\frac{1}{2}\delta^4 y_s-\frac{1}{4}\delta^5 y_{s+\frac{1}{2}}+\dots\right]$$

II інтерполяційний поліном Гаусса дає

$$y(x)=G_n^{II}(x_s+uh)=y_s+u\delta y_{s-\frac{1}{2}}+\frac{u(u+1)}{2!}\delta^2 y_s+\frac{u(u^2-1)}{3!}\delta^3 y_{s-\frac{1}{2}}+$$
$$+\frac{u(u^2-1)(u+2)}{4!}\delta^4 y_s+\dots$$

Послідовно диференціюємо

$$y'(x)=\frac{1}{h}\left[\delta y_{s-\frac{1}{2}}+\frac{2u+1}{2!}\delta^2 y_s+\frac{3u^2-1}{3!}\delta^3 y_{s-\frac{1}{2}}+\frac{4u^3+6u^2-2u-2}{4!}\delta^4 y_s+\right.$$
$$\left.+\frac{5u^4-15u^2+4}{5!}\delta^5 y_{s-\frac{1}{2}}+\frac{6u^5-15u^4-20u^3+45u^2+8u-12}{6!}\delta^6 y_s+\dots\right],$$

$$y''(x)=\frac{1}{h^2}\left[\delta^2 y_s+\frac{6u}{3!}\delta^3 y_{s-\frac{1}{2}}+\frac{12u^2+12u-6}{4!}\delta^4 y_s+\right.$$
$$\left.+\frac{20u^3-30u}{5!}\delta^5 y_{s-\frac{1}{2}}+\frac{30u^4-60u^3-60u^2-90u+8}{6!}\delta^6 y_s+\ldots\right],$$
$$y'''(x)=\frac{1}{h^3}\left[\delta^3 y_{s-\frac{1}{2}}+\frac{24u+12}{4!}\delta^4 y_s+\frac{60u^2-30}{5!}\delta^5 y_{s-\frac{1}{2}}+\right.$$
$$\left.+\frac{120u^3-180u^2-120u+90}{6!}\delta^6 y_s+\ldots\right].$$

При u=0 II інтерполяційний поліном Гаусса дає

$$y'_s=\frac{1}{h}\left[\delta y_{s-\frac{1}{2}}+\frac{1}{2}\delta^2 y_s-\frac{1}{6}\delta^3 y_{s-\frac{1}{2}}+\frac{1}{12}\delta^4 y_s-\ldots\right]$$
$$y''(x)=\frac{1}{h^2}\left[\delta^2 y_s-\frac{1}{12}\delta^4 y_s+\frac{1}{90}\delta^6 y_s-\ldots\right],$$
$$y'''(x)=\frac{1}{h^3}\left[\delta^3 y_{s-\frac{1}{2}}+\frac{1}{2}\delta^4 y_s-\frac{1}{4}\delta^5 y_{s-\frac{1}{2}}+\frac{1}{8}\delta^6 y_s-\ldots\right],$$

і т.д.

Формулу варто порівнювати з формулою параграфа 4.3.

Інтерполяційний поліном Стірлінга

$$y(x)=S_n(x_s+uh)=y_s+u\mu\delta y_s+\frac{u^2}{2!}\delta^2 y_s+\frac{u(u-1)}{3!}\mu\delta^3 y_s+$$
$$+\frac{u^2(u^2-1)}{4!}\delta^4 y_s+\frac{u(u^2-1)(u^2-2^2)}{5!}\mu\delta^5 y_s+\ldots,$$

звідки

$$y'(x)=\frac{1}{h}\left[\mu\delta y_s+u\delta^2 y_s+\frac{3u^2-1}{3!}\mu\delta^3 y_s+\frac{4u^3-2u}{4!}\delta^4 y_s+\right.$$
$$\left.+\frac{5u^4-15u^2+4}{5!}\mu\delta^5 y_s+\frac{6u^5-20u^3+8u}{6!}\delta^6 y_s+\ldots\right],$$
$$y''(x)=\frac{1}{h^2}\left[\delta^2 y_s+\frac{6u}{3!}\mu\delta^3 y_s+\frac{12u^2-2}{4!}\delta^4 y_s+\right.$$
$$\left.+\frac{20u^3-30u}{5!}\mu\delta^5 y_s+\frac{30u^4-60u^2+8}{6!}\delta^6 y_s+\ldots\right],$$
$$y'''(x)=\frac{1}{h^3}\left[\mu\delta^3 y_s+\frac{24u}{4!}\delta^4 y_s+\frac{60u^2-30}{5!}\mu\delta^5 y_s+\frac{120u^3-120u}{6!}\delta^6 y_s+\ldots\right].$$

При u=0 маємо

$$y'_s = \frac{1}{h}\left[\mu\delta y_s - \frac{1}{6}\mu\delta^3 y_s + \frac{1}{30}\mu\delta^5 y_s + \ldots\right],$$

$$y''_s = \frac{1}{h^2}\left[\delta^2 y_s - \frac{\delta^4 y_s}{12} + \frac{\delta^6 y_s}{90} - \ldots\right],$$

$$y'''_s = \frac{1}{h^3}\left[\mu\delta^3 y_s - \frac{1}{4}\mu\delta^5 y_s + \ldots\right],$$

Формули - доцільно порівняти з формулами , , параграфа 4.3. І, нарешті, інтерполяційний поліном Бесселя. Представимо його так

$$y(x) = B_n(x_s + uh) = \frac{y_s + y_{s+1}}{2} + \left(u - \frac{1}{2}\right)\Delta y_s + \frac{u^2 - u}{2} \cdot \frac{\Delta^2 y_{s-1} + \Delta^2 y_s}{2} +$$

$$+ \frac{2u^3 - 2u^2 + u}{12} \cdot \Delta^3 y_{s-1} + \frac{u^4 - 2u^3 - u^2 + 2u}{24} \cdot \frac{\Delta^4 y_{s-2} + \Delta^4 y_{s-1}}{2} + \ldots$$

Диференціюємо його

$$y'(x) = \Delta y_s + \frac{2u-1}{2} \cdot \frac{\Delta^2 y_{s-1} + \Delta^2 y_s}{2} + \frac{6u^2 - 4u + 1}{12} \cdot \Delta^3 y_{s-1} +$$

$$+ \frac{2u^3 - 3u^2 - u + 1}{12} \cdot \frac{\Delta^4 y_{s-2} + \Delta^4 y_{s-1}}{2} + \ldots$$

$$y''(x) = \frac{\Delta^2 y_{s-1} + \Delta^2 y_s}{2} + \frac{12u - 8}{12} \cdot \Delta^3 y_{s-1} +$$

$$+ \frac{6u^2 - 6u - 1}{12} \cdot \frac{\Delta^4 y_{s-2} + \Delta^4 y_{s-1}}{2} + \ldots$$

$$y'''(x) = \Delta^3 y_{s-1} + \frac{2u-1}{2} \cdot \frac{\Delta^4 y_{s-2} + \Delta^4 y_{s-1}}{2} + \ldots$$

Підхід до обчислення інтеграла

$$I = \int_a^b y(x)dx$$

аналогічний підходу до чисельного визначення похідних, а саме: підінтегральна функція замінюється деяким інтерполяційним поліномом, який потім інтегрується в заданих межах.

Нехай це буде, наприклад, І інтерполяційний поліном Ньютона, в якому для зручності подальших обчислень порозкриваємо дужки

$$y(x) = y_s + u \cdot \Delta y_s + \frac{u^2 - u}{2}\Delta^2 y_s + \frac{u^3 - 3u^2 + 2u}{6}\Delta^3 y_s +$$

$$+ \frac{u^4 - 6u^3 + 11u^2 - 6u}{24}\Delta^4 y_s + \frac{u^5 - 10u^4 + 35u^3 - 50u^2 + 24u}{120}\Delta^5 y_s +$$

$$+ \frac{u^6 - 15u^5 + 85u^4 - 225u^3 + 274u^2 - 120u}{720}\Delta^6 y_s + \ldots$$

Інтервал інтегрування розіб'ємо на m однакових кроків $h=(b-a)/m$, де h – крок інтегрування). Візьмемо інтеграл від в межах одного кроку

$$\int_{x_s}^{x_{s+1}} y(x)dx = h\int_0^1 \left[y_s + u\cdot\Delta y_s + \frac{u^2-u}{2}\Delta^2 y_s + \frac{u^3-3u^2+2u}{6}\Delta^3 y_s + \right.$$

$$\left. + \frac{u^4-6u^3+11u^2-6u}{24}\Delta^4 y_s + \ldots \right] du.$$

Після виконання операції інтегрування маємо

$$\int_{x_s}^{x_{s+1}} y(x)dx = h\left(y_s + \frac{1}{2}\Delta y_s - \frac{1}{12}\Delta^2 y_s + \frac{1}{24}\Delta^3 y_s - \frac{19}{720}\Delta^4 y_s + \ldots \right)$$

Отже, для інтеграла в цілому одержуємо

$$\int_a^b y(x)dx = h\sum_{s=0}^{m-1}\left(y_s + \frac{1}{2}\Delta y_s - \frac{1}{12}\Delta^2 y_s + \frac{1}{24}\Delta^3 y_s - \frac{19}{720}\Delta^4 y_s + \ldots \right).$$

Якщо в дужках формули обмежитись лише першим доданком, то

$$\int_a^b y(x)dx \approx h\sum_{s=0}^{m-1} y_s, \quad \text{де} \quad h = \frac{b-a}{m}.$$

Це так звана формула прямокутників.

Інколи формулу записують так

$$\int_a^b y(x)dx \approx h\sum_{s=1}^{m} y_s, \quad \text{де} \quad h = \frac{b-a}{m}.$$

Це – формула правих прямокутників, тоді відповідно формулу можна називати формулою лівих прямокутників.

Якщо в формулі врахувати два перших доданка, то

$$\int_a^b y(x)dx = h\sum_{s=0}^{m-1}\left(y_s + \frac{y_{s+1}-y_s}{2} \right)$$

або

$$\int_a^b y(x)dx \approx \frac{h}{2}\sum_{s=0}^{m-1}(y_s + y_{s+1}), \quad h = \frac{b-a}{m}.$$

Це – формула трапецій.

Її можна записати так

$$\int_a^b y(x)dx = h\left(\frac{y_0+y_m}{2} + \sum_{s=1}^{m-1} y_s \right), \quad \text{де} \quad h = \frac{b-a}{m}.$$

Очевидно, що формули , точніші від формул , . Процес уточнення результату інтегрування шляхом врахування чергових доданків в формулі можна продовжувати, але ми дещо модифікуємо процес. Розіб'ємо інтервал інтегрування на кількість кроків, кратну 2, тобто візьмемо $m=2n$. Підінтегральну функцію замінимо тим же І інтерполяційним поліномом

Ньютона , утримуючи в цьому три перших доданки. Отже, на інтервалі в 2 кроки матимемо

$$\int_{x_s}^{x_{s+2}} y(x)dx = h\int_0^2 \left[y_s + u\cdot\Delta y_s + \frac{u^2-u}{2}\Delta^2 y_s \right] du = \frac{h}{3}\left(y_s + 4y_{s+1} + y_{s+2}\right).$$

Інтеграл в цілому буде

$$\int_a^b y(x)dx = \frac{h}{3}\sum_{s=0}^{n-1}\left(y_{2s} + 4y_{2s+1} + y_{2s+2}\right), \quad h = \frac{b-a}{2n},$$

Це – формула Симпсона.

Якщо інтервал інтегрування розбити на число кроків, кратних 3, тобто взяти $m=3n$, то на ділянці в 3 кроки матимемо

$$\int_{x_s}^{x_{s+3}} y(x)dx \cong h\int_0^3 \left[y_s + u\cdot\Delta y_s + \frac{u^2-u}{2}\Delta^2 y_s + \frac{u^3-3u^2+2u}{6}\Delta^3 y_s \right] du =$$

$$= \frac{3h}{8}\left[y_s + 3\left(y_{s+1} + y_{s+2}\right) + y_{s+3}\right].$$

Тут ми взяли 4 перших члена у формулі .

Інтеграл в цілому

$$\int_a^b y(x)dx = \frac{3h}{8}\sum_{s=0}^{n-1}\left[y_{3s} + 3\left(y_{3s+1} + y_{3s+2}\right) + y_{3s+3}\right], \quad \text{де } h = \frac{b-a}{3n}.$$

Це – формула Ньютона.

Якщо інтервал $[a..b]$ розбивати на $4n$ кроків, то утримуючи в 5 перших членів, інтеграл в межах 4 кроків буде

$$\int_{x_s}^{x_{s+4}} y(x)dx \cong h\int_0^4 \left[y_s + u\cdot\Delta y_s + \frac{u^2-u}{2}\Delta^2 y_s + \frac{u^3-3u^2+2u}{6}\Delta^3 y_s + \right.$$

$$\left. + \frac{u^4-6u^3+11u^2-6u}{24}\Delta^4 y_s \right] du = \frac{2h}{45}\left[7y_s + 32y_{s+1} + 12y_{s+2} + 32y_{s+3} + 7y_{s+4}\right].$$

і інтеграл в цілому

$$\int_a^b y(x)dx = \frac{2h}{45}\sum_{s=0}^{n-1}\left[7y_{4s} + 32y_{4s+1} + 12y_{4s+2} + 32y_{4s+3} + 7y_{4s+4}\right], \quad \text{де} \quad h = \frac{b-a}{4n}.$$

Це – формула Боде.

Розділимо тепер інтервал інтегрування на число кроків, кратне 6 ($m=6n$), а в підінтегральній функції (І інтерполяційний поліном Ньютона) утримаємо перші 7 членів.

На відрізку довжиною 6 кроків матимемо

$$\int_{x_s}^{x_{s+6}} y(x)dx \cong h\int_0^6 \Big[y_s + u\cdot\Delta y_s + \frac{u^2-u}{2}\Delta^2 y_s + \frac{u^3-3u^2+2u}{6}\Delta^3 y_s +$$

$$+\frac{u^4-6u^3+11u^2-6u}{24}\Delta^4 y_s + \frac{u^5-10u^4+35u^3-50u^2+24u}{120}\Delta^5 y_s +$$

$$+\frac{u^6-15u^5+85u^4-225u^3+274u^2-120u}{720}\Delta^6 y_s \Big] du.$$

Остаточно це дорівнює

$$\int_{x_s}^{x_{s+6}} y(x)dx \cong h\cdot\big(6y_s + 18\cdot\Delta y_s + 27\cdot\Delta^2 y_s + 24\cdot\Delta^3 y_s +$$

$$+\frac{123}{10}\Delta^4 y_s + \frac{33}{10}\Delta^5 y_s + \frac{41}{140}\Delta^6 y_s \Big] du.$$

Замінимо в правій частині останній коефіцієнт $\frac{41}{140}$ на $\frac{3}{10}$ (похибки буде $\frac{42}{140}-\frac{41}{140}=\frac{1}{140}$). Тоді остання формула приводиться до вигляду

$$\int_{x_s}^{x_{s+6}} y(x)dx \cong \frac{3h}{10}\cdot\big(y_s + 5y_{s+1} + y_{s+2} + 6y_{s+3} +$$

$$+ y_{s+4} + 5y_{s+5} + y_{s+6}\big)$$

і інтеграл в цілому

$$\int_a^b y(x)dx = \frac{3h}{10}\sum_{s=0}^{n-1}\big[y_{6s} + y_{6s+2} + y_{6s+4} + y_{6s+6} +$$

$$+5\big(y_{6s+1}+y_{6s+5}\big)+6y_{6s+3}\big], \quad \text{де } h=\frac{b-a}{6n}.$$

Це – формула Уеддля.

Можна показати, що наведені формули мають такі порядки точності:
формула прямокутників ρ=1, формула трапецій ρ=2,
формула Симпсона ρ=3, формула Ньютона ρ=4,
формула Боде ρ=5, формула Уеддля ρ=6.

Лекція 16. Метод Ейлера та його модифікації для системи рівнянь 1-го порядку

Ідею методу Ейлера розглянемо спочатку стосовно до диференціального рівняння .

Наближений розв'язок диференціального рівняння будемо шукати на рівномірній сітці значень *t*, а саме на сітці

$$t_\nu = \nu\tau,$$

де $\nu = 0,1,2,3,...$ - номер кроку,

τ - величина кроку за часом *t*.

Змінна y при $t_\nu = \nu\tau$ позначається так:

$$y\Big|_{t=\nu\tau} = y_\nu .$$

Замінимо *dy/dt* в наближено йому відповідним різницевим співвідношенням виду

$$\frac{dy}{dt}\Big|_{t=\nu\tau} \approx \frac{y_{\nu+1} - y_\nu}{\tau}.$$

Підставляємо в та розв'язуємо одержане рівняння відносно $y_{\nu+1}$:

$$y_{\nu+1} = y_\nu + \tau \cdot f(y_\nu, \nu\tau).$$

Формулу можна розглядати, як рекурентну. Якщо відоме початкове значення y_0, то при $\nu = 1$ з визначається:

$$y_1 = y_0 + \tau \cdot f(y_0, 0).$$

Після цього з при $\nu = 2$ маємо:

$$y_2 = y_1 + \tau \cdot f(y_1, \tau)$$

і так далі.

Формула є формальним записом методу Ейлера розв'язання задачі Коші для рівняння .

Стосовно до системи диференціальних рівнянь в канонічній формі крок за часом за методом Ейлера буде

$$y_{z,\nu+1} = y_{z,\nu} + \tau \cdot (\sum_{s=1}^{n} A_{z,s} y_{s,\nu} + \sum_{s=1}^{m} b_{z,s} x_{s,\nu}),\ 1 \le z \le n .$$

Формулу реалізує підпрограма *EilSu*:

```
procedure          EilSu(A,B:Coef2;Mx0:Coef;Tau:Real;var
Y:Coef);
var z,s,n,m: integer;
    Sum: real;
    Yn:Coef;
begin
  n:=round(A[1,-1]); m:=round(B[1,-1]);
  for z:=1 to n do
    begin
      Sum:=0;
      for s:=1 to n do Sum:=Sum+A[z,s]*Y[s];
      for s:=1 to m do Sum:=Sum+B[z,s]*Mx0[s];
      Yn[z]:=Y[z]+Tau*Sum;
    end;
  Yn[-1]:=n; Y:=Yn;
end;
```

Тут *A,B* - масиви коефіцієнтів системи;

*Mx*0 - масив значень вхідних сигналів при $t=\nu\tau$ (за станом на початку кроку за часом);

Tau - величина кроку;

Y - масив параметрів стану (виходів досліджуваного об'єкта), одержується процедурою за станом на початок кроку, повертається - за станом на кінець кроку.

Крок за часом для системи заміщення:

$$U_{z,\nu+1} = U_{z,\nu} + \tau \cdot (k_z x_\nu + U_{z+1,\nu}),\ \ 1 \le z \le n,$$

де

$$U_{n+1,\nu} = -\frac{1}{a_n}\sum_{s=0}^{n-1} a_s U_{s+1,\nu}\ .$$

Реалізує його підпрограма *EilWp*.

```
procedure EilWp(A,K:Coef;x0,x1,Tau:real; var U:Coef);
var z,s,n: integer;
    Sum: real;
begin
  n:=round(A[-1]); Sum:=0;
  for s:=0 to n-1 do
    Sum:=Sum+A[s]*U[s+1];
  U[n+1]:=-Sum/A[n];
  for z:=1 to n do
    U[z]:=U[z]+Tau*(K[z]+x0*U[z+1]);
  U[0]:= K[0]*x1+U[1]
end;
```

Тут *A,K* - масиви коефіцієнтів лівої частини диференціального рівняння та системи заміщення;

*x*0, *x*1 - значення вхідного сигналу на початку та в кінці кроку;

Tau - величина кроку за часом;

U - масив змінних стану системи заміщення (передається програмі за станом на початок кроку, повертається нею - за станом на кінець кроку). Вихід об'єкта - в комірці *U*[0].

Перша та друга модифікації методу Ейлера

Графічно алгоритм Ейлера - це лінійна екстраполяція розв'язку на крок вперед.

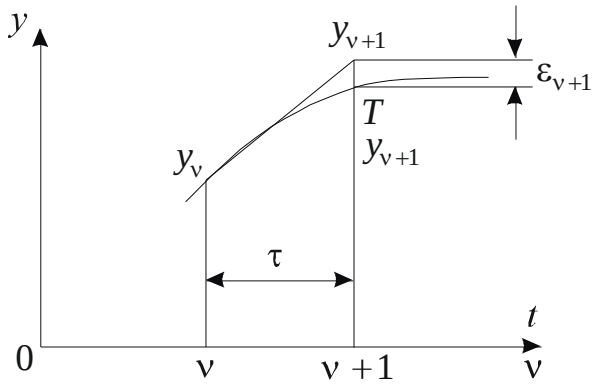

Рис.11. Крок за часом в методі Ейлера

Якщо крива на рис.11 являє собою точний розв'язок, отже, y_ν - точний розв'язок при $t=\nu\tau$, то $\tilde{y}_{\nu+1}$, одержане в результаті виконання кроку методом Ейлера, буде відрізнятись від точного $y_{\nu+1}$ на величину $\varepsilon_{\nu+1}$:

$$\varepsilon_{\nu+1}=\tilde{y}_{\nu+1}-y_{\nu+1}.$$

Очевидно, що $|\varepsilon_{\nu+1}|$ буде тим більшим, чим більша кривизна кривої *y(t)* в околі точки з номером ν ($t=\nu\tau$) та чим більше величина кроку τ.

Можна спробувати зменшити $|\varepsilon_{\nu+1}|$, якщо визначати функцію *f(y,t)* не на початку, а в середині виконуваного кроку. Для цього можна спочатку методом Ейлера виконати допоміжний крок величиною $\frac{\tau}{2}$, а потім уже, маючи наближене значення $y_{\nu+0.5}$, виконати робочий крок від точки з номером ν до точки з номером $\nu+1$. Отже, для окремо взятого диференціального рівняння матимемо

$$\begin{cases} y_{\nu+1/2}=y_\nu+\dfrac{\tau}{2}f(y_\nu,\nu\tau); \\ y_{\nu+1}=y_\nu+\tau f(y_{\nu+1/2},(\nu+1/2)\tau). \end{cases}$$

Система реалізує так званий 1-ий модифікований метод Ейлера.

Для системи канонічних диференціальних рівнянь це буде:

$$\begin{cases} y_{z,\nu+1/2}=y_{z,\nu}+\dfrac{\tau}{2}\left(\sum\limits_{s=1}^{n}A_{z,s}y_{s,\nu}+\sum\limits_{s=1}^{m}B_{z,s}x_{s,\nu}\right), & 1\le z\le n; \\ y_{z,\nu+1}=y_{z,\nu}+\tau\left(\sum\limits_{s=1}^{n}A_{z,s}y_{s,\nu+1/2}+\sum\limits_{s=1}^{m}B_{z,s}x_{s,\nu+1/2}\right), & 1\le z\le n. \end{cases}$$

Алгоритм реалізується в підпрограмі *Eil1Su*.

```
procedure Eil1Su(A,B:Coef2;Mx0,MxS:Coef;Tau:Real;
                 var Y:Coef);
var z,s,n,m: integer;
    Sum: real;
    Yp: Coef;
begin
  n:=round(A[1,-1]); m:=round(B[1,-1]);
  Yp:=Y; EilSu(A,B,Mx0,v0/2,Yp);
```

```
   for z:=1 to n do
    begin
      Sum:=0;
      for s:=1 to n do Sum:=Sum+A[z,s]*Yp[s];
      for s:=1 to m do Sum:=Sum+B[z,s]*MxS[s];
      Y[z] := Y[z]+Tau*Sum
    end
end;
```

Тут *Mx0*, *MxS* - масиви значень входів при $t=\nu\tau$ та $t=(\nu+1)\tau$.

Y - масив параметрів стану системи (передається за станом на $t=n\tau$, повертається - за станом на $t=(\nu+1)\tau$).

Для системи заміщення I модифікований метод Ейлера дає

$$\begin{cases} U_{z,\nu+1/2} = U_{z,\nu} + \dfrac{\tau}{2}(k_z x_\nu + U_{z+1,\nu}), 1 \le z \le n, \\ \partial e \\ U_{n+1,\nu} = -\dfrac{1}{a_n}\sum\limits_{s=0}^{n-1} a_s U_{s+1,\nu}; \\ U_{z,\nu+1} = U_{z,\nu} + \tau(k_z x_{\nu+1/2} + U_{z+1,\nu+1/2}), 1 \le z \le n, \\ \partial e \\ U_{n+1,\nu+1/2} = -\dfrac{1}{a_n}\sum\limits_{s=0}^{n-1} a_s U_{s+1,\nu+1/2}. \end{cases}$$

Алгоритм реалізується в підпрограмі *Eil1Wp*.

```
procedure Eil1Wp(A,K:Coef;x0,xs,x1,Tau:real;
                 var U:Coef);
var s,n: integer;
    Sum: real;
    Up: Coef;
begin
  n:=round(A[-1]); Up:=U; EilWp(A,K,x0,x1,Tau/2,Up);
  Sum:=0;
  for s:=0 to n-1 do Sum:=Sum+A[s]*Up[s+1];
  Up[n+1]:=-Sum/A[n];
  for s:=1 to n do
    U[s] := U[s] + Tau*(k[s]*xs+Up[s+1]);
  U[0] := k[0]*x1+u[1]
end;
```

Тут *A, K* - як звичайно масиви коефіцієнтів лівої частини диференціального рівняння та системи заміщення;

*x*0, *xs*, *x*1 - значення вхідного сигналу на початку, в середині та в кінці кроку;

Tau - величина кроку;

Y - масив параметрів стану системи заміщення (передається за станом на початок кроку, повертається - за станом на кінець кроку).

У 2-му модифікованому методі Ейлера спочатку якимось методом виконується перший крок, а потім процес розв'язання реалізується подвійними кроками, з $(\nu\text{-}1)$ -го до $(\nu+1)$-го. Для окремого диференціального рівняння в канонічній формі це виглядає так

$$y_{\nu+1} = y_{\nu-1} + 2\tau \cdot f(y_\nu, \nu\tau).$$

Для системи канонічних рівнянь маємо

$$y_{z,\nu+1} = y_{z,\nu-1} + 2\tau\left(\sum_{s=1}^{n} A_{z,s} y_{z,\nu} + \sum_{s=1}^{m} B_{z,s} x_{s,\nu}\right).$$

Відповідно маємо процедуру *Eil2Su*:

```
Procedure Eil2Su(A,B:Coef2;Mx0:Coef;v0:real;
                                        var Yp,Y:Coef);
var z,s,n,m:integer;
    v2,sum:real;
    y1: Coef;
begin
  n:=round(A[1,-1]); m:=round(B[1,-1]); v2:=2*v0;
  for z := 1 to n do
    begin
      Sum:=0;
      for s:=1 to n do Sum:=Sum+A[z,s]*Y[s];
      for s:=1 to m do Sum:=Sum+B[z,s]*Mx0[s];
      Y1[z]:=Yp[z]+v2*Sum
    end;
  Yp:=Y;Y:=Y1;
end;
```

Тут *Mx0* - масив значень вхідного сигналу на початку кроку;

Yp,Y - масиви значень виходів системи на моменти часу $t=(\nu\text{-}1)\tau$ та $t=\nu\tau$.

Для системи заміщення крок за часом 2-им модифікованим методом Ейлера

$$y_{z,\nu+1} = y_{\nu-1} + 2\tau(k_z x_\nu + y_{z+1,\nu}), 1 \le z \le n,$$

де

$$y_{n+1,\nu} = -\frac{1}{a_n}\sum_{s=0}^{n-1} a_s y_{s+1,\nu}.$$

Реалізує цей алгоритм підпрограма *Eil2Wp*.

```
procedure Eil2Wp(A,K:Coef;x0,x1,v0:real;var Yp,Y:Coef);
var s,n: integer;
    v2,sum: real;
    y1: Coef;
```

```
begin
  n:=round(A[-1]); v2:=2*v0; Sum:=0;
  for s:=0 to n-1 do Sum:=Sum+A[s]*Y[s+1];
  Y[n+1]:=-Sum/A[n];
  for s:=1 to n do
    y1[s]:=Yp[s]+v2*(k[s]*x0+Y[s+1]);
  y1[0]:=k[0]*x1+y1[1]; Yp:=Y;Y:=y1
end;
```

Досвід експлуатації 2-го модифікованого методу Ейлера показує, що цей алгоритм є нестійким. Його реалізація супроводжується швидким наростанням модуля похибки, похибка носить коливний характер і наростає лавиноподібно.

При необхідності послідовного виконання значної кількості кроків метод не можна рекомендувати.

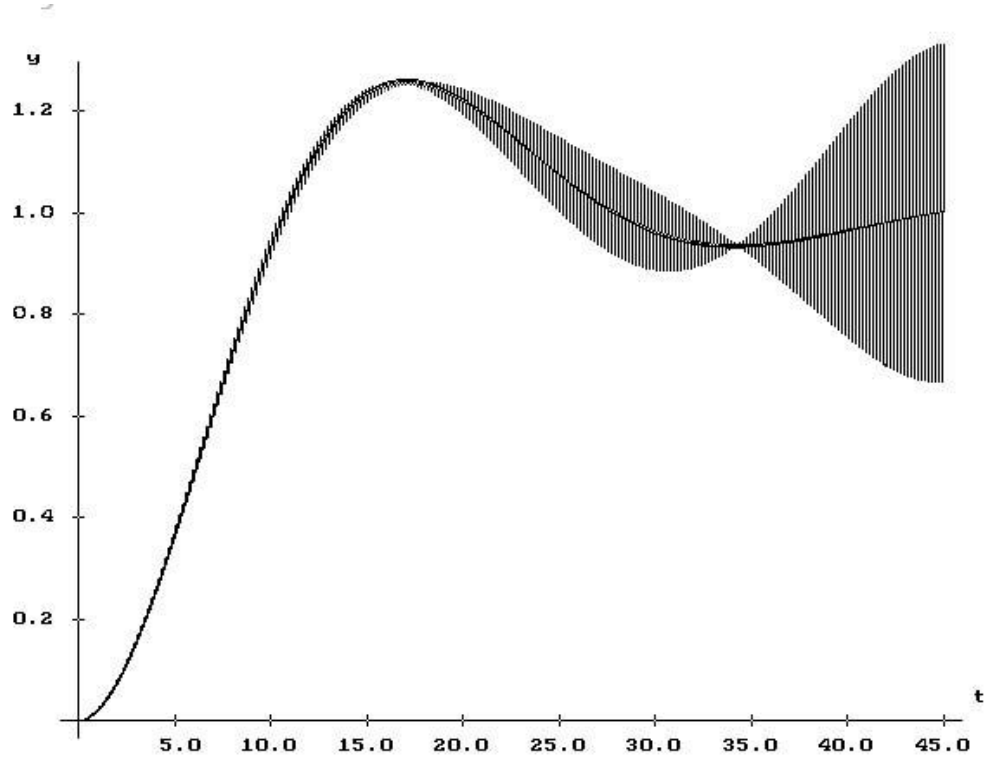

Рис.12. Розв’язок диференціального рівняння 2-им модифікованим методом

Лекція 17. Метод Рунге-Кутта для системи канонічних рівнянь

Метод Рунге-Кутта-Мерсона - інтегрування диференціальних рівнянь та їх систем крок інтегрування τ вважався фіксованим, тобто не змінювався в процесі інтегрування. Проте очевидно, що в залежності від характеру зміни вхідних сигналів, а також від темпу та характеру зміни параметрів стану, які проявляються в процесі інтегрування, крок інтегрування мав би якимось чином залежати від згаданих чинників. Автоматичний вибір кроку інтегрування (корекція його в процесі інтегрування) реалізується, зокрема, в

методі Рунге-Кутта-Мерсона, до розгляду якого ми зараз і переходимо. Пропонується така розрахункова схема

$$\begin{cases} R_1 = \tau \cdot f(y_v, t_v); \\ R_2 = \tau \cdot f(y_v + \frac{1}{3} R_1, t_v + \frac{\tau}{3}); \\ R_3 = \tau \cdot f(y_v + \frac{R_1 + R_2}{6}, t_v + \frac{\tau}{3}); \\ R_4 = \tau \cdot f(y_v + \frac{R_1 + 3R_3}{8}, t_v + \frac{\tau}{2}); \\ R_5 = \tau \cdot f(y_v + \frac{R_1 - 3R_3 + 4R_4}{2}, t_v + \tau); \\ y_{v+1} = y_v + \frac{R_1 + 4R_4 + R_5}{6}; \\ \varepsilon_{v+1} = \frac{-2R_1 + 9R_2 - 8R_4 + R_5}{30}, \ \rho = 5. \end{cases}$$

Якщо не рахувати останнього співвідношення, то схема нічим принципово не відрізняється від раніше розглянутих схем методу Рунге-Кутта. Принциповим є якраз останнє співвідношення, яке визначає похибку на кроці. Ця похибка співставляється з допустимою ε_d похибкою і якщо

$$\varepsilon_{v+1} \le \frac{\varepsilon_d}{30},$$

то крок інтегрування τ збільшується вдвічі, якщо ж

$$\varepsilon_{v+1} > \varepsilon_d,$$

то крок інтегрування зменшується вдвічі, якщо ж похибка ε_{v+1} вкладається в діапазон

$$\frac{\varepsilon_d}{30} \le \varepsilon_{v+1} \le \varepsilon_d,$$

то крок інтегрування вважається прийнятним, отже, він залишається без зміни. Для системи лінійних диференціальних рівнянь алгоритм з коригуванням кроку за умовами - можна представити так

$$
\begin{cases}
R_{z,1} = \tau\left(\sum\limits_{s=1}^{n} A_{z,s} y_{s,v} + \sum\limits_{s=1}^{m} B_{z,s} x_{s,v} \right), \quad 1 \le z \le n; \\
R_{z,2} = \tau\left(\sum\limits_{s=1}^{n} A_{z,s} \left(y_{s,v} + \frac{1}{3} R_{s,1} \right) + \sum\limits_{s=1}^{m} B_{z,s} x_{s,v+\frac{1}{3}} \right), \quad 1 \le z \le n; \\
R_{z,3} = \tau\left(\sum\limits_{s=1}^{n} A_{z,s} \left(y_{s,v} + \frac{R_{s,1} + R_{s,2}}{6} \right) + \sum\limits_{s=1}^{m} B_{z,s} x_{s,v+\frac{1}{3}} \right), \quad 1 \le z \le n; \\
R_{z,4} = \tau\left(\sum\limits_{s=1}^{n} A_{z,s} \left(y_{s,v} + \frac{R_{s,1} + 3R_{s,3}}{8} \right) + \sum\limits_{s=1}^{m} B_{z,s} x_{s,v+\frac{1}{2}} \right), \quad 1 \le z \le n; \\
R_{z,5} = \tau\left(\sum\limits_{s=1}^{n} A_{z,s} \left(y_{s,v} + \frac{R_{s,1} - 3R_{s,3} + 4R_{s,4}}{2} \right) + \sum\limits_{s=1}^{m} B_{z,s} x_{s,v+1} \right), \quad 1 \le z \le n; \\
y_{z,v+1} = y_{z,v} + \frac{R_{z,1} + 4R_{z,4} + R_{z,5}}{6}, \quad 1 \le z \le n; \\
\varepsilon_{z,v+1} = \frac{-2R_{z,1} + 9R_{z,2} - 8R_{z,4} + R_{z,5}}{30}, \quad 1 \le z \le n, \quad \rho = 5.
\end{cases}
$$

Цю систему співвідношень можна переписати так:

$$
\begin{cases}
R_{z,1} = \tau\left(C_z + \sum\limits_{s=1}^{m} B_{z,s} x_{s,v} \right), \quad 1 \le z \le n; \\
R_{z,2} = \tau\left(C_z + \frac{1}{3} \sum\limits_{s=1}^{n} A_{z,s} R_{s,1} + \sum\limits_{s=1}^{m} B_{z,s} x_{s,v+\frac{1}{3}} \right), \quad 1 \le z \le n; \\
R_{z,3} = \tau\left(C_z + \frac{1}{6} \sum\limits_{s=1}^{n} A_{z,s} \left(R_{s,1} + R_{s,2} \right) + \sum\limits_{s=1}^{m} B_{z,s} x_{s,v+\frac{1}{3}} \right), \quad 1 \le z \le n; \\
R_{z,4} = \tau\left(C_z + \frac{1}{8} \sum\limits_{s=1}^{n} A_{z,s} \left(R_{s,1} + 3R_{s,3} \right) + \sum\limits_{s=1}^{m} B_{z,s} x_{s,v+\frac{1}{2}} \right), \quad 1 \le z \le n; \\
R_{z,5} = \tau\left(C_z + \frac{1}{2} \sum\limits_{s=1}^{n} A_{z,s} \left(R_{s,1} - 3R_{s,3} + 4R_{s,4} \right) + \sum\limits_{s=1}^{m} B_{z,s} x_{s,v+1} \right), \quad 1 \le z \le n; \\
y_{z,v+1} = y_{z,v} + \frac{R_{z,1} + 4R_{z,4} + R_{z,5}}{6}, \quad 1 \le z \le n; \\
\varepsilon_{z,v+1} = \frac{-2R_{z,1} + 9R_{z,2} - 8R_{z,4} + R_{z,5}}{30}, \quad 1 \le z \le n,
\end{cases}
$$

де

$$C_z = \sum_{s=1}^{n} A_{z,s} y_{s,v}, \quad 1 \le z \le n.$$

Алгоритм з виконанням ітерацій, обумовлених аналізом похибки на кроці, реалізує підпрограма *MersSu*.

```
procedure MersSu(A,B:Coef2; Xn,Kx,Epsd:Coef;
                   var t,Tau:real; var Y:Coef);
var w,z,s,n,m: integer;
    R: array[1..30,1..4] of real;
    Eps,sa,sb: real;
    Ug,Us: boolean;
begin
  n:=round(A[1,-1]); m:=round(B[1,-1]);
  for z:=1 to n do
    begin
      C[z]:=0; sb:=0;
      for  s:=1 to n do
        C[z]:=C[z]+A[z,s]*Y[s];
      for s:=1 to m do
        sb:=sb*B[z,s]*Xn[s];
      R[z,1]:=Tau*(C[z]+sb)
    end;
  repeat
    for w:=2 to 4 do
      begin
        case w of
          2,3 :dt:=Tau/3;
            4:dt:= Tau/2;
            4:dt:= Tau;
        end;
        for z:=1 to n do
          begin
            sa:=0;
            for s:=1 to n do
              case w of
                2: sa:=sa+A[z,s]*R[s,1]/3;
                3: sa:=sa+A[z,s]*R[s,2]/6;
                4: sa:=sa+A[z,s]*(R[s,1]+3*R[s,3])/8;
                4: sa:=sa+A[z,s]*(R[s,1]-
                          3*R[s,3]+R[s,4])/2
              end
              for s:=1 to m do
                sa:=sa+B[z,s]*(Xn[s]+Kx[s]*dt);
              R[z,w]:=Tau*(C[z]+sa)
```

```
          end
      end;
    Ug:=true; Us:=false;
    for z:=1 to n do
      begin
        Eps:=(-2*R[z,1]+9*R[z,2]-8*R[z,4]+R[z,4])/30;
        if Eps>Epsd[z]/30 then Ug:=false;
        if Eps>Epsd[z] then Us:=true
      end;
    if Ug then Tau:=2*Tau;
          else
            if Us
              then Tau:=Tau/2
              else
                begin
                  for z:=1 to n do

U[z]:=U[z]+(R[z,1]+4*R[z,4]+R[z,4])/6;
                    t:=t+Tau
                end
  until (Ug=false) and (Us=false)
end;
```

Тут *Xn* - масив значень *x* на початку кроку,

Kx - масив значень похідних за *t* від *x* на початку кроку,

t - значення часу на початку кроку.

Таким чином, в підпрограмі реалізовано гіпотезу про лінійну екстраполяцію вхідних сигналів в межах кроку.

Для системи заміщення - алгоритм матиме такий вигляд:

$$
\begin{cases}
U_{n+1,v} = -\dfrac{1}{a_n}\sum\limits_{s=0}^{n-1} a_s U_{s+1,v}; \\
R_{z,1} = \tau\left(k_z x_v + U_{z+1,v}\right), & 1 \le z \le n; \\
R_{z,2} = \tau\left(k_z x_{v+\frac{1}{3}} + U_{z+1,v} + \dfrac{R_{z,1}}{3}\right), & 1 \le z \le n; \\
R_{z,3} = \tau\left(k_z x_{v+\frac{1}{3}} + U_{z+1,v} + \dfrac{R_{z,2}}{2}\right), & 1 \le z \le n; \\
R_{z,4} = \tau\left(k_z x_{v+\frac{1}{2}} + U_{z+1,v} + \dfrac{R_{z,1}+3R_{z,3}}{8}\right), & 1 \le z \le n; \\
R_{z,5} = \tau\left(k_z x_{v+1} + U_{z+1,v} + \dfrac{R_{z,1}-3R_{z,3}+4R_{z,4}}{2}\right), & 1 \le z \le n; \\
y_{z,v+1} = y_{z,v} + \dfrac{R_{z+1}+4R_{z,4}+R_{z,5}}{6}, & 1 \le z \le n; \\
U_{0,v+1} = k_0 x_{v+1} + U_{1,v+1}; \\
\varepsilon_{z,v+1} = \dfrac{-2R_{z,1}+9R_{z,2}-8R_{z,4}+R_{z,5}}{30}, & 1 \le z \le n, \quad \rho = 5.
\end{cases}
$$

Реалізуємо алгоритм в підпрограмі *MersWp*

```
procedure MersWp(A,K,Epsd:Coef;xn,xk: real;
                    var t,Tau:real;var U:Coef);
var n,s,z:integer;
    Ug,Us:boolean;
    r1,r2,r3,r4,r4,Sum,Eps:real;
begin
  Sum:=0; n:=round(A[-1]);
  for s:=0 to n-1 do
    Sum:=Sum+A[s]*U[s+1];
  U[n+1]:=-Sum/A[n];
  repeat
    Ug:=true; Us:=false;
    for z:=1 to n do
      begin
        r1:=Tau*(K[z]*xn+U[z+1]);
        r2:=Tau*(K[z]*(xn+kx*Tau/3)+U[z+1]+r1/3);
        r3:=Tau*(K[z]*(xn+kx*Tau/3)+U[z+1]+r2/2);

r4:=Tau*(K[z]*(xn+kx*Tau/2)+U[z+1]+(r1+3*r3)/8);
        r4:=Tau*(K[z]*(xn+kx*Tau)+U[z+1]+(r1-
```

```
3*r3+4*r4)/2);
        Eps:=(-2*r1+9*r2-8*r4+r4)/30;
        if Eps>Epsd[z]/30 then Ug:=false;
        if Eps>Epsd[z] then Us:=true
      end;
    if Ug then Tau:=2*Tau
          else if Us then Tau:=Tau/2
                     else
                       begin
                         for z:=1 to n do
                           U[z]:=U[z]+(r1+4*r4+r4)/6;
                         t:=t+Tau
                       end
  until (Ug=false) and (Us=false)
end;
```

Метод Рунге-Кутта-Фельберга - було запропоновано ряд алгоритмів числового інтегрування звичайних диференціальних рівнянь з автоматичною корекцією кроку різних порядків точності.

Умови коригування кроку дещо відрізняються від аналогічних умов в методі Мерсона, а саме замість умови маємо:

якщо

$$\varepsilon_{v+1} \le \frac{\varepsilon_d}{20},$$

то крок інтегрування збільшується вдвічі, якщо ж

$$\varepsilon_{v+1} > \varepsilon_d,$$

то крок зменшується вдвічі.

Отже, умова залишається такою ж, як і в методі Рунге-Кутта-Мерсона.

Наведемо нижче розрахункові формули, запропоновані Фельбергом для різних ρ.

$\rho = 1$

$$\begin{cases} R_1 = \tau \cdot f\left(y_v, t_v\right); \\ R_2 = \tau \cdot f\left(y_v + \dfrac{1}{2} R_1, t_v + \dfrac{\tau}{2}\right); \\ R_3 = \tau \cdot f\left(y_v + \dfrac{R_1 + 255R_2}{256}, t_v + \tau\right); \\ y_{v+1} = y_v + \dfrac{R_1 + 510R_2 + R_3}{512}; \\ \varepsilon_{v+1} = \dfrac{R_1 - R_3}{512}. \end{cases}$$

$\rho = 2$

$$\begin{cases} R_1 = \tau \cdot f\left(y_v, t_v\right); \\ R_2 = \tau \cdot f\left(y_v + \dfrac{R_1}{4} R_1, t_v + \dfrac{\tau}{4}\right); \\ R_3 = \tau \cdot f\left(y_v + \dfrac{-189R_1 + 729R_2}{800}, t_v + \dfrac{27}{40}\tau\right); \\ R_4 = \tau \cdot f\left(y_v + \dfrac{214R_1 + 27R_2 + 650R_3}{891}, t_v + \tau\right); \\ y_{v+1} = y_v + \dfrac{533R_1 + 1600R_3 - 27R_4}{2106}; \\ \varepsilon_{v+1} = \dfrac{-299R_1 + 702R_2 - 700R_3 + 297R_4}{23166}. \end{cases}$$

$\rho = 3$

$$\begin{cases} R_1 = \tau \cdot f\left(y_v, t_v\right); \\ R_2 = \tau \cdot f\left(y_v + \dfrac{R_1}{4}, t_v + \dfrac{\tau}{4}\right); \\ R_3 = \tau \cdot f\left(y_v + \dfrac{4R_1 + 32R_2}{81}, t_v + \dfrac{4}{9}\tau\right); \\ R_4 = \tau \cdot f\left(y_v + \dfrac{399R_1 - 864R_2 + 1053R_3}{686}, t_v + \dfrac{6}{7}\tau\right); \\ R_5 = \tau \cdot f\left(y_v + \dfrac{26R_1 + 81R_3 + 49R_4}{156}, t_v + \tau\right); \\ y_{v+1} = y_v + \dfrac{559R_1 + 2187R_3 + 686R_4 + 312R_5}{3744}; \\ \varepsilon_{v+1} \dfrac{65R_1 - 243R_3 + 490R_4 - 312R_5}{3744}. \end{cases}$$

$\rho = 4$

$$\begin{cases} R_1 = \tau \cdot f\left(y_v, t_v\right); \\ R_2 = \tau \cdot f\left(y_v + \dfrac{2}{9}R_1, t_v + \dfrac{2}{9}\tau\right); \\ R_3 = \tau \cdot f\left(y_v + \dfrac{R_1 + 3R_2}{12}, t_v + \dfrac{1}{3}\tau\right); \\ R_4 = \tau \cdot f\left(y_v + \dfrac{69R_1 - 243R_2 + 270R_3}{128}, t_v + \dfrac{3}{4}\tau\right); \\ R_5 = \tau \cdot f\left(y_v + \dfrac{-85R_1 + 405R_2 - 324R_3 + 64R_4}{60}, t_v + \tau\right); \\ R_6 = \tau \cdot f\left(y_v + \dfrac{65R_1 - 135R_2 + 351R_3 + 64R_4 + 15R_5}{432}, t_v + \dfrac{5}{6}\tau\right); \\ y_{v+1} = y_v + \dfrac{47R_1 + 216R_3 + 15R_5 + 108R_6}{8448}; \\ \varepsilon_{v+1} = \dfrac{2R_1 - 9R_3 + 64R_4 + 15R_5 - 72R_6}{300}. \end{cases}$$

$\rho = 5$

$$
\begin{cases}
R_1 = \tau \cdot f\left(y_v, t_v\right); \\
R_2 = \tau \cdot f\left(y_v + \dfrac{R_1}{6}, t_v + \dfrac{\tau}{6}\right); \\
R_3 = \tau \cdot f\left(y_v + \dfrac{4R_1 + 16R_2}{75}, t_v + \dfrac{4}{15}\tau\right); \\
R_4 = \tau \cdot f\left(y_v + \dfrac{20R_1 - 9R_2 + 5R_3}{24}, t_v + \dfrac{2}{3}\tau\right); \\
R_5 = \tau \cdot f\left(y_v + \dfrac{-40R_1 + 144R_2 - 100R_3 + 16R_4}{25}, t_v + \dfrac{4}{5}\tau\right); \\
R_6 = \tau \cdot f\left(y_v + \dfrac{722R_1 - 2304R_2 + 2035R_3 - 88R_4 + 275R_5}{640}, t_v + \tau\right); \\
R_7 = \tau \cdot f\left(y_v + \dfrac{-154R_1 + 385R_3 - 616R_4 + 385R_5}{8960}, t_v\right); \\
R_8 = \tau \cdot f\left(y_v + \dfrac{3906R_1 - 14336R_2 + 84315R_3 - 84280R_4}{26880} + \right. \\
\qquad\qquad \left. + \dfrac{10395R_5 + 26880R_7}{26880}, t_v + \tau\right); \\
y_{v+1} = y_v + \dfrac{42R_1 + 3375R_3 + 2376R_4 + 1375R_5 + 640R_7 + 640R_8}{8448}; \\
\varepsilon_{v+1} = \dfrac{5\left(R_1 + R_6 - R_7 - R_8\right)}{66}.
\end{cases}
$$

$\rho = 7$

$$
\begin{cases}
R_1 = \tau \cdot f\left(y_v, t_v\right); \\
R_2 = \tau \cdot f\left(y_v + \dfrac{2}{27} R_1, t_v + \dfrac{2}{27}\tau\right); \\
R_3 = \tau \cdot f\left(y_v + \dfrac{R_1 + 3R_2}{36}, t_v + \dfrac{1}{9}\tau\right); \\
R_4 = \tau \cdot f\left(y_v + \dfrac{R_1 + 3R_3}{24}, t_v + \dfrac{1}{6}\tau\right); \\
R_5 = \tau \cdot f\left(y_v + \dfrac{20R_1 - 75R_3 + 75R_4}{48}, t_v + \dfrac{5}{12}\tau\right) \\
R_6 = \tau \cdot f\left(y_v + \dfrac{R_1 + 5R_4 + 4R_5}{20}, t_v + \dfrac{1}{2}\tau\right); \\
R_7 = \tau \cdot f\left(y_v + \dfrac{-25R_1 + 125R_4 - 260R_5 + 250R_6}{108}, t_v + \dfrac{5}{6}\tau\right); \\
R_8 = \tau \cdot f\left(y_v + \dfrac{93R_1 + 244R_4 - 100R_6 + 13R_7}{900}, t_v + \dfrac{1}{6}\tau\right); \\
R_9 = \tau \cdot f\left(y_v + \dfrac{180R_1 - 795R_4 + 1408R_5 - 1070R_6 + 67R_7 + 270R_8}{90}, t_v + \dfrac{2}{3}\tau\right); \\
R_{10} = \tau \cdot f\left(y_v + \dfrac{-455R_1 + 115R_4 - 3904R_5 + 3110R_6 - 171R_7 + 1530R_8 - 45R_9}{540}, t_v + \dfrac{\tau}{3}\right); \\
R_{11} = \tau \cdot f\left(\begin{array}{l} y_v + \dfrac{2383R_1 - 8525R_4 + 17984R_5 - 15050R_6}{4100} + \\ + \dfrac{2133R_7 + 2250R_8 + 1125R_9 + 1800R_{10}}{4100}, t_v + \tau \end{array}\right); \\
R_{12} = \tau \cdot f\left(y_v + \dfrac{3R_1 - 30R_6 - 3R_7 - 15R_8 + 15R_9 + 30R_{10}}{205}, t_v\right); \\
R_{13} = \tau \cdot f\left(\begin{array}{l} y_v + \dfrac{-1777R_1 - 8525R_4 + 17984R_5 - 14450R_6 + 2193R_7}{4100} + \\ + \dfrac{2550R_8 + 825R_9 + 1200R_{10} + 4100R_{12}}{4100}, t_v + \tau \end{array}\right); \\
y_{v+1} = y_v + \dfrac{272R_1 + 216(R_7 + R_8) + 27(R_9 + R_{10}) + 41(R_{11} + R_{12})}{840}; \\
\varepsilon_{v+1} = \dfrac{41(R_1 + R_{11} - R_{12} - R_{13})}{840}; \rho = 7.
\end{cases}
$$

Програмна реалізація методів Фельберга практично не відрізняється від такої для методу Мерсона, тому ми тут її не розглядаємо. Читач може зробити це самостійно. Якщо вхідні сигнали в системі канонічних рівнянь

змінюються так, що лінійна екстраполяція їх в межах кроку інтегрування вважається неприйнятною, можна скористатись відповідними підпрограмами (глобальними по відношенню до процедури, що реалізує крок) або передати відповідні підпрограми в процедуру, що реалізує крок, як параметри.

Лекція 18. Явна, неявна та комбінована схеми інтегрування одновимірного рівняння теплопровідності

Необхідність враховувати (і розраховувати) динаміку температури в об'єктах, що описуються диференціальним рівнянням , виникає при дослідженні температурних полів в теплоізолюючих матеріалах та конструкціях, таких як стіни, обмуровка, обмазка, футеровка, так і при врахуванні впливу подібних конструктивних елементів на динаміку печей, реакторів в хімічній, харчовій, металургійній промисловості, в промисловості будівельних матеріалів та конструкцій і в багатьох інших випадках.
В той же час числове інтегрування диференціального рівняння можна розглядати як модельну задачу для відпрацювання алгоритмів ефективного розв'язання задач з просторовим розподіленням параметрів, оскільки, з одного боку, рівняння історично було одним з перших, з якого почалось дослідження динаміки об'єктів з розподіленими параметрами, і уже з цієї причини на сьогоднішній день накопичено великий досвід, а, з іншого боку, для рівняння існують аналітичні розв'язки, з якими можна співставляти результати числового інтегрування, оцінюючи їх коректність та ефективність.
Переходимо до дискретних змінних

$$\begin{cases} x = sh = x_s \ ; \\ t = \nu\tau = t_\nu , \end{cases}$$

де s,ν - номери кроків за x та t,
h,τ - величини кроків відповідно.

Крок h стосовно до розгляданої задачі будемо визначати як

$$h = \frac{\delta}{m},$$

де δ - товщина стінки;
m - ціле додатнє число (число шарів, на які розбивається стінка по товщині в цілях розрахунку).

Крок за часом в подальшому будемо визначати як

$$\tau = \frac{Dt}{Ks},$$

де, в свою чергу,

$$Dt = \frac{D}{L}.$$

Тут D - заданий час спостереження процесу;

L - ширина графіка зміни температури в функції часу в пікселях (ціле додатнє число ≈ 500);

Ks- ціле додатнє число (кратність подрібнення кроку за часом, коли крок Dt вважається недопустимо великим).

Значення температури в точці, де $x = sh$, а $t = \nu\tau$

$$\theta\Big|_{\substack{x=sh\\t=\nu\tau}} = \theta_{s,\nu}.$$

Розкладемо значення температури θ в околі точки $x = sh$, $t = \nu\tau$ в ряд Тейлора відносно t

$$\theta\Big|_{\substack{x=sh\\t=\nu\tau+\Delta t}} = \theta\Big|_{\substack{x=sh\\t=\nu\tau}} + \frac{\partial\theta}{\partial t}\Big|_{\substack{x=sh\\t=\nu\tau}} \cdot \Delta t + \frac{1}{2!}\frac{\partial^2\theta}{\partial t^2}\Big|_{\substack{x=sh\\t=\nu\tau}} \cdot (\Delta t)^2 + \frac{1}{3!}\frac{\partial^3\theta}{\partial t^3}\Big|_{\substack{x=sh\\t=\nu\tau}} \cdot (\Delta t)^3 + \ldots$$

Тоді при $\Delta t = \tau$ матимемо

$$\theta\Big|_{\substack{x=sh\\t=\nu\tau+\tau}} = \theta\Big|_{t=\nu\tau} + \frac{\partial\theta}{\partial t}\Big|_{\substack{x=sh\\t=\nu\tau}} \tau + \frac{1}{2!}\frac{\partial^2\theta}{\partial t^2}\Big|_{\substack{x=sh\\t=\nu\tau}} \cdot \tau^2 + \frac{1}{3!}\frac{\partial^3\theta}{\partial t^3}\Big|_{\substack{x=sh\\t=\nu\tau}} \cdot \tau^3 + \ldots$$

Розв'язуємо відносно $\left.\frac{\partial\theta}{\partial t}\right|_{\substack{x=sh\\t=\nu\tau}}$

$$\frac{\partial\theta}{\partial t}\Big|_{\substack{x=sh\\t=\nu\tau}} = \frac{\theta_{s,\nu+1} - \theta_{s,\nu}}{\tau} + R_t,$$

де

$$R_t = -\left\{\frac{1}{2}\frac{\partial^2\theta}{\partial t^2}\Big|_{\substack{x=sh\\t=\nu\tau}} \tau + \frac{1}{6}\frac{\partial^3\theta}{\partial t^3}\Big|_{\substack{x=sh\\t=\nu\tau}} \tau^2 + \ldots\right\}.$$

Тепер розглянемо θ в околі тієї ж точки $x = sh$, $t = \nu\tau$, але на цей раз відносно x

$$\theta\Big|_{\substack{x=sh+\Delta x\\t=\nu\tau}} = \theta\Big|_{\substack{x=sh\\t=\nu\tau}} + \frac{\partial\theta}{\partial x}\Big|_{\substack{x=sh\\t=\nu\tau}} \Delta x + \frac{1}{2}\frac{\partial^2\theta}{\partial x^2}\Big|_{\substack{x=sh\\t=\nu\tau}} \cdot (\Delta x)^2 + \frac{1}{6}\frac{\partial^3\theta}{\partial x^3}\Big|_{\substack{x=sh\\t=\nu\tau}} \cdot (\Delta x)^3 + \frac{1}{24}\frac{\partial^4\theta}{\partial x^4}\Big|_{\substack{x=sh\\t=\nu\tau}} \cdot (\Delta x)^4 + \ldots$$

При $\Delta x = h$ з маємо

$$\theta\Big|_{\substack{x=(s+1)h\\t=\nu\tau}} = \theta\Big|_{\substack{x=sh\\t=\nu\tau}} + \frac{\partial\theta}{\partial x}\Big|_{\substack{x=sh\\t=\nu\tau}} h + \frac{1}{2}\frac{\partial^2\theta}{\partial x^2}\Big|_{\substack{x=sh\\t=\nu\tau}} \cdot h^2 +$$

$$+\frac{1}{6}\frac{\partial^3\theta}{\partial x^3}\Big|_{\substack{x=sh\\t=\nu\tau}} \cdot h^3 + \frac{1}{24}\frac{\partial^4\theta}{\partial x^4}\Big|_{\substack{x=sh\\t=\nu\tau}} \cdot h^4 + \ldots$$

При $\Delta x = -h$ з одержуємо

$$\theta\Big|_{\substack{x=(s-1)h\\t=\nu\tau}} = \theta\Big|_{\substack{x=sh\\t=\nu\tau}} - \frac{\partial\theta}{\partial x}\Big|_{\substack{x=sh\\t=\nu\tau}} h + \frac{1}{2}\frac{\partial^2\theta}{\partial x^2}\Big|_{\substack{x=sh\\t=\nu\tau}} \cdot h^2 -$$

$$-\frac{1}{6}\frac{\partial^3\theta}{\partial x^3}\Big|_{\substack{x=sh\\t=\nu\tau}} \cdot h^3 + \frac{1}{24}\frac{\partial^4\theta}{\partial x^4}\Big|_{\substack{x=sh\\t=\nu\tau}} \cdot h^4 - \ldots$$

Додаємо та

$$\theta_{s+1,\nu} + \theta_{s-1,\nu} = 2\theta_{s,\nu} + \frac{\partial^2\theta}{\partial x^2}\Big|_{\substack{x=sh\\t=\nu\tau}} \cdot h^2 + \frac{1}{12}\frac{\partial^4\theta}{\partial x^4}\Big|_{\substack{x=sh\\t=\nu\tau}} \cdot h^4 + \ldots$$

Розв'яжемо тепер відносно $\frac{\partial^2\theta}{\partial x^2}\Big|_{\substack{x=sh\\t=\nu\tau}}$

$$\frac{\partial^2\theta}{\partial x^2}\Big|_{\substack{x=sh\\t=\nu\tau}} = \frac{\theta_{s-1,\nu} - 2\theta_{s,\nu} + \theta_{s+1,\nu}}{h^2} + R_{x^2},$$

де

$$R_{x^2} = -\frac{1}{12}\frac{\partial^4\theta}{\partial x^4}\Big|_{\substack{x=sh\\t=\nu\tau}} h^2 - \frac{1}{360}\frac{\partial^6\theta}{\partial x^6}\Big|_{\substack{x=sh\\t=\nu\tau}} h^4 - \ldots$$

Підставимо тепер $\frac{\partial\theta}{\partial t}\Big|_{\substack{x=sh\\t=\nu\tau}}$ з та $\frac{\partial^2\theta}{\partial x^2}\Big|_{\substack{x=sh\\t=\nu\tau}}$ з у, що розглядається в околі точки $x = sh$, $t = \nu\tau$. Матимемо

$$\frac{\theta_{s,\nu+1} - \theta_{s,\nu}}{\tau} = a\frac{\theta_{s-1,\nu} - 2\theta_{s,\nu} + \theta_{s+1,\nu}}{h^2} + R_{tx^2},$$

де

$$R_{tx^2} = \frac{1}{2}\frac{\partial^2\theta}{\partial t^2}\Big|_{\substack{x=sh\\t=\nu\tau}} \tau + \frac{1}{6}\frac{\partial^3\theta}{\partial t^3}\Big|_{\substack{x=sh\\t=\nu\tau}} \tau^2 + \ldots -$$

$$-a\left[\frac{1}{12}\frac{\partial^4\theta}{\partial x^4}\Big|_{\substack{x=sh\\t=\nu\tau}} \tau^2 + \frac{1}{360}\frac{\partial^6\theta}{\partial x^6}\Big|_{\substack{x=sh\\t=\nu\tau}} \tau^4 + \ldots\right].$$

Якщо в відкинути залишковий член R_{tx^2}, та розв'язати одержане співвідношення відносно $\theta_{s,\nu+1}$, то матимемо

$$\theta_{s,\nu+1} = \theta_{s,\nu} + \frac{a\tau}{h^2}\left(\theta_{s-1,\nu} - 2\theta_{s,\nu} + \theta_{s+1,\nu}\right).$$

Вводимо позначення

$$\begin{cases} q = \dfrac{a\tau}{h^2}; \\ q_1 = 1 - 2q. \end{cases}$$

Тепер приводиться до вигляду

$$\theta_{s,\nu+1} = q_1\theta_{s,\nu} + q\left(\theta_{s-1,\nu} + \theta_{s+1,\nu}\right), \qquad 1 \le s \le m-1.$$

Формула задає так звану явну схему числового інтегрування рівняння теплопровідності .

Очевидно, що формула може застосовуватись лише в діапазоні $1 \le s \le m-1$, тобто для "обслуговування" внутрішніх вузлів "сітки", з її допомогою можна оновити (перейти від $t = \nu\tau$ до $t = (\nu+1)\tau$) значення температури всередині тіла. Що ж стосується границь тіла (поверхонь $x = 0$ та $x = \delta$), то для оновлення температур на поверхнях треба звертатись до граничних умов. Простіше всього коли це відповідно *Ngl*=1 та (чи) *Ngr*=1 - формули , .

Ngl=1, $\theta\big|_{0,\nu+1} = T_l\left((\nu+1)\tau\right);$

Ngr=1, $\theta\big|_{m,\nu+1} = T_r\left((\nu+1)\tau\right).$

При *Ngl*=2 матимемо, згідно з

$$-\lambda\frac{\theta_{1,\nu+1} - \theta_{0,\nu+1}}{h} = Q_l\left((\nu+1)\tau\right),$$

звідки

$$\theta_{0,\nu+1} = \theta_{1,\nu+1} + \frac{h}{\lambda}Q_l\left((\nu+1)\tau\right)$$

і при *Ngr*=2 згідно з

$$-\lambda\frac{\theta_{m,\nu+1} - \theta_{m-1,\nu+1}}{h} = Q_r\left((\nu+1)\tau\right),$$

звідки

$$\theta_{m,\nu+1} = \theta_{m-1,\nu+1} - \frac{h}{\lambda}Q_r\left((\nu+1)\tau\right).$$

При *Ngl*=3 - формула

$$-\lambda\frac{\theta_{1,\nu+1} - \theta_{0,\nu+1}}{h} = \alpha_0\left(T_{sl}\left((\nu+1)\tau\right) - \theta_{0,\nu+1}\right),$$

звідки

$$\theta_{0,\nu+1} = b_0\theta_{1,\nu+1} + b_1T_{sl}\left((\nu+1)\tau\right),$$

де

$$\begin{cases} b_0 = \dfrac{\lambda}{\lambda + h\alpha_0}; \\ b_1 = \dfrac{h\alpha_0}{\lambda + h\alpha_0} = 1 - b_0. \end{cases}$$

При *Ngr*=3 - формула

$$-\lambda\frac{\theta_{m,\nu+1}-\theta_{m-1,\nu+1}}{h}=\alpha_1\left(\theta_{m,\nu+1}-T_{sr}((\nu+1)\tau\right),$$

звідки

$$\theta_{m,\nu+1}=b_2\theta_{m-1,\nu+1}+b_3T_{sr}((\nu+1)\tau),$$

де

$$\begin{cases} b_2=\dfrac{\lambda}{\lambda+h\alpha_1}; \\ b_3=\dfrac{h\alpha_1}{\lambda+h\alpha_1}=1-b_2. \end{cases}$$

Якщо дискретним аналогом поточного ($t=\nu\tau$) температурного поля взяти масив *Teta : CoefR*, то крок за часом з використанням формул - можна оформити у вигляді процедури *Stept*.

```
procedure Stept(var Teta:CoefR);
var s:integer;
    Tet:CoefR;
begin
  t := t+Tau;
  for s:=1 to m-1 do
    Tet[s]:=q1*Teta[s]+q*(Teta[s-1]+Teta[s+1]);
  case Ngl of
    1:Tet[0]:=Tl(t);
    2:Tet[0]:=Tet[1]+Hx/Lam*Ql(t);
    3:Tet[0]:=b0*Tet[1]+b1*Tsl(t)
  end
  case Ngr of
    1:Tet[m]:=Tr(t);
    2:Tet[m]:=Tet[m-1]-Hx/Lam*Qr(t);
    3:Tet[m]:=b2*Tet[m-1]+b3*Tsr(t)
  end;
  Tet[-1]:=m; Teta:=Tet
end;
```

Тут використовуються глобальні змінні *t* - час, *Tau* – крок τ, *Hx* - крок *h*, *Lam* - коефіцієнт теплопровідності λ; A0, A1 - коефіцієнти тепловіддачі α_0 та α_1, коефіцієнти b_0, b_1, b_2, b_3 - формули , . *Tl,Tr,Ql,Qr,Tsl,Tsr* - ідентифікатори підпрограм-функцій.
Масив *Teta* передається процедурі за станом на $t=\nu\tau$, а повертається нею за станом $t=(\nu+1)\tau$.

Для числового інтегрування диференціального рівняння за умови, що $\frac{\partial^2\theta}{\partial\varphi^2}=\frac{\partial^2\theta}{\partial z^2}=0$ матимемо

$$\frac{\partial\theta}{\partial t}=a\left(\frac{\partial^2\theta}{\partial r^2}+\frac{1}{r}\frac{\partial\theta}{\partial r}\right).$$

Замінимо похідні таким чином

$$\frac{\theta_{s,\nu+1}-\theta_{s,\nu}}{\tau}=a\left(\frac{\theta_{s-1,\nu}-2\theta_{s,\nu}+\theta_{s+1,\nu}}{h^2}+\frac{1}{r_0+sh}\cdot\frac{\theta_{s+1,\nu}-\theta_{s-1,\nu}}{2h}\right).$$

Тут

$r=r_0+sh;$

$h=(r_1-r_0)/m.$

r_0 - внутрішній радіус циліндра,

r_1 - зовнішній радіус.

Розв'яжемо відносно $\theta_{s,\nu+1}$

$$\theta_{s,\nu+1}=q_1\theta_{s+1,\nu}+q\left[(1-r_s)\theta_{s-1,\nu}+(1+r_s)\theta_{s+1,\nu}\right],$$

де

$$r_s=\frac{1}{2\left(\frac{r_0}{h}+s\right)}.$$

Граничні умови для циліндра позначимо так

Ngv - для внутрішньої поверхні; *Ngz* - для зовнішньої.

Тоді

Ngv=1, $\theta_{0,\nu+1}=T_\nu((\nu+1)\tau);$

Ngz=1, $\theta_{m,\nu+1}=T_z((\nu+1)\tau);$

Ngv=2, $-\lambda\frac{\theta_{1,\nu+1}-\theta_{0,\nu+1}}{h}=Q_\nu((\nu+1)\tau),$

звідки

$$\theta_{0,\nu+1}=\theta_{1,\nu+1}+\frac{h}{\lambda}Q_\nu((\nu+1)\tau).$$

Ngz=2, $-\lambda\frac{\theta_{m,\nu+1}-\theta_{m-1,\nu}}{h}=Q_z((\nu+1)\tau),$

звідки

$$\theta_{m,\nu+1}=\theta_{m-1,\nu+1}-\frac{h}{\lambda}Q_z((\nu+1)\tau).$$

Ngv=3, $-\lambda\frac{\theta_{1,\nu+1}-\theta_{0,\nu+1}}{h}=\alpha_0\left(T_{sv}((\nu+1)\tau)-\theta_{0,\nu+1}\right),$

звідки

$$\theta_{0,\nu+1} = b_0\theta_{1,\nu+1} + b_1 T_{s\nu}\left((\nu+1)\tau\right),$$

де

$$\begin{cases} b_0 = \dfrac{\lambda}{\lambda + h\alpha_0}\ ; \\ b_1 = \dfrac{h\alpha_0}{\lambda + h\alpha_0} = 1 - b_0. \end{cases}$$

Ngz=3, $$-\lambda\frac{\theta_{m,\nu+1} - \theta_{m-1,\nu+1}}{h} = \alpha_1\left(\theta_{m,\nu+1} - T_{sz}\left((\nu+1)\tau\right)\right),$$

звідки

$$\theta_{m,\nu+1} = b_2\theta_{m-1,\nu+1} + b_3 T_{sz}\left((\nu+1)\tau\right),$$

де

$$\begin{cases} b_2 = \dfrac{\lambda}{\lambda + h\alpha_1}\ ; \\ b_3 = \dfrac{h\alpha_1}{\lambda + h\alpha_1} = 1 - b_2. \end{cases}$$

Тут, як і для плоскої стінки

α_0, α_1-коефіцієнти тепловіддачі на внутрішній та на зовнішній поверхнях циліндра.

Щоб вивести розрахункове співвідношення для осі суцільного циліндра, розглянемо спочатку

$$\lim_{r\to 0}\frac{1}{r}\cdot\frac{\partial\theta}{\partial r} = \lim_{r\to 0}\frac{\dfrac{\partial\theta}{\partial r}}{r}.$$

Розкриваємо невизначеність за правилом Лопіталя

$$\lim_{r\to 0}\frac{1}{r}\cdot\frac{\partial\theta}{\partial r} = \lim_{r\to 0}\frac{\partial^2\theta}{\partial r^2}.$$

Одержане співвідношення підставляємо в при $r \to 0$. Одержуємо

$$\left.\frac{\partial\theta}{\partial r}\right|_{r\to 0} = 2a\left.\frac{\partial^2\theta}{\partial r^2}\right|_{r\to 0}.$$

У дискретній формі

$$\frac{\theta_{0,\nu+1} - \theta_{0,\nu}}{\tau} = 2a\frac{\theta_{-1,\nu} - 2\theta_{0,\nu} + \theta_{1,\nu}}{h^2}.$$

Але ж за умови температурної симетрії $\theta_{-1,\nu} = \theta_{1,\nu}$, отже

$$\frac{\theta_{0,\nu+1} - \theta_{0,\nu}}{\tau} = 4a\frac{\theta_{1,\nu} - \theta_{0,\nu}}{h^2},$$

звідки

$\theta_{0,\nu+1} = \theta_{0,\nu} + 4q(\theta_{1,\nu} - \theta_{0,\nu})$

або

$\theta_{0,v+1} = (1-4q)\theta_{0,v} + 4q\theta_{1,v}$.

Крок за часом для циліндра - підпрограма *SteptC*.

```
procedure SteptC(var Teta:CoefR);
var s: integer;
    Rs: real;
    Tet: CoefR;
begin
  for s:=1 to m-1 do
    begin
      Rs:=1/(2*(r0/Hx+s);
      Tet[s]=q1*Teta[s]+q*((1-Rs)*Teta[s-1]+
             (1+Rs)*Teta[s+1])
    end;
  t:=t+Tau;
  if r0=0
    then Tet[0]:=(1-4*q)*Teta[0]+4*q*Teta[1]
    else
      case Ngv of
        1: Tet[0]:=Tv(t);
        2: Tet[0]:=Tet[1]+Hx/Lam*Qv(t);
        3: Tet[0]:=b0*Tet[1]+b1*Tsv(t)
      end;
      case Ngz of
        1: Tet[m]:=Tz(t);
        2:Tet[m]:=Tet[m-1]-Hx/Lam*Qz(t);
        3: Tet[m]:=b2*Tet[m-1]+b3*Tsz(t)
      end;
  Tet[-1]:=m; Teta:=Tet;
end;
```

Для кулі згідно з

$$\frac{\theta_{s,v+1} - \theta_{s,v}}{\tau} = a\left(\frac{\theta_{s-1,v} - 2\theta_{s,v} + \theta_{s+1,v}}{h^2} + \frac{2}{r_0 + sh} \cdot \frac{\theta_{s+1,v} - \theta_{s-1,v}}{2h}\right).$$

Тут, як і для циліндра

$$h = \frac{r_1 - r_0}{m};$$

$$r = r_0 + sh,$$

де r_0, r_1 - внутрішній та зовнішній радіуси кулі.

Розв'язуємо відносно $\theta_{s,v+1}$

$$\theta_{s,v+1} = \theta_{s,v} + q\left(\theta_{s-1,v} - 2\theta_{s,v} + \theta_{s+1,v} + \frac{1}{\frac{r_0}{h} + S} \cdot \left(\theta_{s+1,v} - \theta_{s-1,v}\right)\right)$$

або

$$\theta_{s,v+1} = q\theta_{s,v} + q\left[(1 - r_s)\theta_{s-1,v} + (1 + r_s)\theta_{s+1,v}\right],$$

де

$$r_s = \frac{1}{\frac{r_0}{h} + S}.$$

Граничні умови для кулі з $r_0 \neq 0$ ті ж, що для пустотілого циліндра. Для центра сфери

$$\lim_{r \to 0} \frac{2}{r} \cdot \frac{\partial \theta}{\partial r} = 2\frac{\partial^2 \theta}{\partial r^2}.$$

Диференціальне рівняння теплопровідності для суцільної кулі

$$\left.\frac{\partial \theta}{\partial t}\right|_{r=0} = 3a\frac{\partial^2 \theta}{\partial r^2}.$$

Замінюємо похідні різницевими співвідношеннями

$$\frac{\theta_{0,v+1} - \theta_{0,v}}{\tau} = 3a\frac{\theta_{-1,v} - 2\theta_{0,v} + \theta_{1,v}}{h^2}.$$

Враховуючи, що з умови симетрії $\theta_{-1,v} = \theta_{1,v}$, маємо

$$\theta_{0,v+1} = \theta_{0,v} + 6q\left(\theta_{1,v} - \theta_{0,v}\right)$$

або

$$\theta_{0,v+1} = (1 - 6q)\theta_{0,v} + 6q\theta_{1,v}.$$

Крок за часом для кулі.

```
procedure SteptS(var Teta:CoefR);
var s: integer;
    Rs: real;
    Tet: CoefR;
begin
  for s:=1 to m-1 do
    begin
      Rs:=1/(r0/Hx+S);
      Tet[s]:=q1*Teta[s]+q*((1-Rs)*Teta[s-1]+
              (1+Rs)*Teta[s+1]);
      t:=t+Tau;
      if r0=0
        then Tet[0]:=(1-6*q)*Teta[0]+6*q*Teta[1]
        else
          case Ngv of
            1:Tet[0]:=Tv(t);
```

```
            2:Tet[0]:=Tet[1]+Hx/Lam*Qv(t);
            3:Tet[0]:=b0*Tet[1]+b1*Tsv(t)
          end;
      case Ngz of
        1:Tet[m]:=Tz(t);
        2: Tet[m]:=Tet[m-1]-Hx/Lam*Qz(t);
        3:Tet[m]:=b2*Tet[m-1]+b3*Tsz(t)
      end;
  Tet[-1]:=m;Teta:=Tet
end;
```

Необхідність виконувати умову стійкості приводить до того, що при фіксованому *h* крок за часом τ не можна брати більшим, ніж це дозволяється умовою

$$\tau \le \frac{h^2}{6a}.$$

Якщо ж диференціальне рівняння треба проінтегрувати на значному інтервалі часу 0..*D*, то кількість необхідних кроків за часом може обчислюватись сотнями, а то і тисячами, а це може бути відчутним не лише при ручному розрахунку, але і для комп'ютера (справжні проблеми для комп'ютера починаються при спробах інтегрувати дво-, а тим більше, тривимірні рівняння типу , тощо). Тому дослідники шукали алгоритм, схему числового інтегрування, яка б не мала обмеження на використання більших кроків τ, а, отже, і більших значень *q*.

Замінимо в похідні різницевими співвідношеннями таким чином

$$\frac{\theta_{s,v+1}-\theta_{s,v}}{\tau} = a\frac{\theta_{s-1,v+1}-2\theta_{s,v+1}+\theta_{s+1,v+1}}{h^2},\ 1 \le s \le m$$

можна переписати так

$$-q\theta_{s-1,v+1}+(1+2q)\theta_{s,v+1}-q\theta_{s+1,v+1}=\theta_{s,v},\ 1 \le s \le m-1.$$

Запис можна розглядати як систему лінійних алгебраїчних рівнянь відносно невідомих $\theta_{s-1,v+1}$, $\theta_{s,v+1}$, $\theta_{s+1,v+1}$, $1 \le s \le m-1$. Усього невідомих *(m*+1*)*, а рівнянь *(m*-1*)*. Отже, не вистачає двох рівнянь. Щоб "доукомплектувати" систему до рівнянь треба додати два рівняння граничних умов. Домовимось, що першим рівнянням буде рівняння граничної умови на "лівій" поверхні стінки (при *x=0*), а останнім - рівняння граничної умови при $x=\delta$.

За такої умови система буде тридіагональною, а розв'язувати ми її будемо методом прогонки (див. наприклад, розділ про інтерполяційні сплайни).

В процесі прямого ходу будемо приводити *s*-те рівняння системи до вигляду

$$\theta_{s,v+1} = A_s\theta_{s+1,v+1} + B_s.$$

Підставимо в замість $\theta_{s-1,v+1}$. Одержимо

$$-q\left(A_{s-1}\theta_{s,v+1}+B_{s-1}\right)+(1+2q)\theta_{s,v+1}-q\theta_{s+1,v+1}=\theta_{s,v},$$

звідки

$$\theta_{s,v+1} = \frac{q}{1+2q-qA_{s-1}} \cdot \theta_{s+1,v+1} + \frac{\theta_{s,v} + qB_{s-1}}{1+2q-qA_{s-1}}.$$

Порівнюємо тепер з . Отримуємо

$$\begin{cases} A_s = \dfrac{q}{1+2q-qA_{s-1}}; \\ B_s = \dfrac{\theta_{s,v} + qB_{s-1}}{1+2q-qA_{s-1}}. \end{cases}$$

Формули можна розглядати як рекурентні. Але щоб ці формули "запрацювали", треба спочатку визначити A_0 та B_0. Для цього скористаємось граничною умовою при *x=0*.

Ngl=1, $\theta_{0,v+1} = T_l\left((v+1)\tau\right).$

Порівнюючи останній вираз з бачимо, що

Ngl=1, $\begin{cases} A_0 = 0; \\ B_0 = T_l\left((v+1)\tau\right). \end{cases}$

Ngl=2, $$-\lambda \frac{\partial\theta}{\partial x}\bigg|_{\substack{x=0 \\ t=(v+1)\tau}} = Q_l\left((v+1)\tau\right)$$

або в дискретній формі

$$-\lambda \frac{\theta_{1,v+1} - \theta_{0,v+1}}{h} = Q_l\left((v+1)\tau\right),$$

або

$$\theta_{0,v+1} = \theta_{1,v+1} + \frac{h}{\lambda} \cdot Q_l\left((v+1)\tau\right),$$

звідки після порівняння з одержуємо

Ngl=2, $\begin{cases} A_0 = 1; \\ B_0 = \dfrac{h}{\lambda} \cdot Q_l\left((v+1)\tau\right). \end{cases}$

І, нарешті,

Ngl=3, $$-\lambda \cdot \frac{\partial\theta}{\partial x}\bigg|_{\substack{x=\delta \\ t=(v+1)\tau}} = \alpha_0\left(T_{sl}\left((v+1)\tau\right) - \theta_{0,v+1}\right)$$

або в дискретній формі

$$-\lambda \frac{\theta_{1,v+1} - \theta_{0,v+1}}{h} = \alpha_0\left(T_{sl}\left((v+1)\tau\right) - \theta_{0,v+1}\right).$$

Розв'язуємо останній вираз відносно $\theta_{0,v+1}$

$$\theta_{0,v+1} = b_0\theta_{1,v+1} + b_1 T_{sl}\left((v+1)\tau\right).$$

Порівнюємо цей вираз з

Ngl=3, $\begin{cases} A_0 = b_0; \\ B_0 = b_1 \cdot T_{sl}\left((v+1)\tau\right). \end{cases}$

Маючи A_0, B_0 в залежності від *Ngl*, розраховуємо за формулами A_s, B_s, $1 \le s \le m-1$. Цим завершується прямий хід в методі прогонки.

Тепер для початку зворотнього ходу треба визначити $\theta_{m,v+1}$.

Ngr=1, $\theta_{m,v+1} = T_r\left((v+1)\tau\right).$

Ngr=2, $-\lambda \cdot \left.\frac{\partial\theta}{\partial x}\right|_{x=\delta} = Q_r\left((v+1)\tau\right).$

В дискретній формі

$$-\lambda \cdot \frac{\theta_{m,v+1} - \theta_{m-1,v+1}}{h} = Q_r\left((v+1)\tau\right)$$

або

$$\theta_{m,v+1} = \theta_{m-1,v+1} - \frac{h}{\lambda} \cdot Q_r\left((v+1)\tau\right).$$

Щоб позбавитись від невідомого $\theta_{m-1,v+1}$ запишемо при *s*=*m*-1 (коефіцієнти A_{m-1} та B_{m-1} на момент закінчення прямого ходу уже визначені).

$$\theta_{m-1,v+1} = A_{m-1}\theta_{m,v+1} + B_{m-1}.$$

Підставимо $\theta_{m-1,v+1}$ з в

$$\theta_{m,v+1} = A_{m-1}\theta_{m,v+1} + B_{m-1} - \frac{h}{\lambda} \cdot Q_r\left((v+1)\tau\right),$$

звідки

Ngr=2, $\theta_{m,v+1} = \dfrac{B_{m-1} - \frac{h}{\lambda} \cdot Q_r\left((v+1)\tau\right)}{1 - A_{m-1}}$

І, нарешті, згідно з формулою

Ngr=3, $\theta_{m,v+1} = b_2\theta_{m-1,v+1} + b_3 T_{sr}\left((v+1)\tau\right).$

Підставимо сюди $\theta_{m-1,v+1}$ з

$$\theta_{m,v+1} = b_2\left(A_{m-1}\theta_{m,v+1} + B_{m-1}\right) + b_3 T_{sr}\left((v+1)\tau\right),$$

звідки

Ngr=3, $\theta_{m,v+1} = \dfrac{b_2 B_{m-1} + b_3 T_{sr}\left((v+1)\tau\right)}{1 - b_2 A_{m-1}}.$

Після того, як $\theta_{m,v+1}$ визначено за формулами , чи в залежності від *Ngr*, за формулою , змінюючи в ній послідовно *s* від *m*-1 до 0, визначаємо усі цікавлячі нас температури, тобто одержуємо розв'язок системи (з добавленими до неї граничними умовами).

Описаний алгоритм виконання кроку за часом реалізується в підпрограмі *StepN*.

```
procedure StepN(var Teta:CoefR);
var s:integer;
    Zn:real;
    A,B:CoefR;
begin
```

```
  t:=t+Tau;
  case Ngl of
    1:begin
        A[0]:=0; B[0]:=Tl(t)
      end;
    2:begin
        A[0]:=1; B[0]:=Hx/Lam*Ql(t)
      end;
    3:begin
        A[0]:=B0; B[0]:=B1*Tsl(t)
      end
  end;
  for s:=1 to m-1 do
    begin
      Zn:=1+q*(2-A[s-1]);
      A[s]:=q/Zn;
      B[s]:=(Teta[s]+q*B[s-1])/Zn
    end;
  case Ngr of
    1:Teta[m]:=Tr(t);
    2:Teta[m]:=(B[m-1]-Hx/Lam*Qr(t))/(1-A[m-1]);
    3:Teta[m]:=(B2*B[m-1]+B3*Tsr(t))/(1-B2*A[m-1])
  end;
  for s:=m-1 downto 0 do
    Teta[s]:=A[s]*Teta[s+1]+B[s]
end;
```

Можна показати, що неявна схема інтегрування диференціального рівняння теплопровідності є стійкою при будь-яких значеннях $q = \frac{a\tau}{h^2}$. Але не варто зловживати свободою вибору q. Завищення значення q (тобто вибір надто великого кроку τ) приведе до накопичення похибок, яке не буде катастрофічним (до втрати стійкості справа не дійде), але суттєве спотворення результату може мати місце. Отже, триматись в околі $q = \frac{1}{6}$ і тут не зайве.

Розглянемо тепер так звану комбіновану схему числового інтегрування диференціального рівняння , або, як її ще називають, схему Кранка-Нікольсона.

Для цього запишемо дискретні аналоги рівняння для явної та неявної схем.

$$\frac{\theta_{s,v+1} - \theta_{s,v}}{\tau} = a \cdot \frac{\theta_{s-1,v} - 2\theta_{s,v} + \theta_{s+1,v}}{h^2};$$

$$\frac{\theta_{s,v+1} - \theta_{s,v}}{\tau} = a \cdot \frac{\theta_{s-1,v} - 2\theta_{s,v+1} + \theta_{s+1,v+1}}{h^2}$$

та знайдемо їх середнє арифметичне

$$\theta_{s,v+1}-\theta_{s,v}=\frac{a\tau}{2h^2}\left(\theta_{s-1,v}-2\theta_{s,v}+\theta_{s+1,v}+\theta_{s-1,v+1}-2\theta_{s,v+1}+\theta_{s+1,v+1}\right).$$

Введемо позначення

$$q_k=\frac{q}{2}.$$

Тоді попереднє співвідношення приводиться до вигляду

$$-q_k\theta_{s-1,v+1}+(1+q)\theta_{s,v+1}-q_k\theta_{s+1,v+1}=\theta_{s,v}+q_k\left(\theta_{s-1,v}-2\theta_{s,v}+\theta_{s+1,v}\right),1\le s\le m-1.$$

Формула якраз і реалізує схему Кранка-Нікольсона. Вона "обслуговує" внутрішні вузли сітки. Зовнішні вузли обслуговують граничні умови. Для них ця схема нічим не відрізнятиметься від неявної. Як і в неявній схемі система алгебраїчних рівнянь є тридіагональною, і для її розв'язання використовується той же метод прогонки. Отже, в процесі виконання прямого ходу, як і в неявній схемі, рівняння приводиться до виду .

Підставимо в замість $\theta_{s-1,v+1}$

$$-q_k\left(A_{s-1}\theta_{s-1,v+1}+B_{s-1}\right)+(1+q)\theta_{s,v+1}-q_k\theta_{s+1,v+1}=\theta_{s,v}+q_k\left(\theta_{s-1,v}-2\theta_{s,v}+\theta_{s+1,v}\right),$$
$$1\le s\le m-1,$$

звідки

$$\theta_{s,v+1}=\frac{q_k}{1+q-q_kA_{s-1}}\cdot\theta_{s+1,v+1}+\frac{\theta_{s,v}+q_k\left(\theta_{s-1,v}-2\theta_{s,v}+\theta_{s+1,v}\right)+q_kB_{s-1}}{1+q-q_kA_{s-1}}.$$

Порівнюємо з . Одержуємо

$$\begin{cases}A_s=\dfrac{q_k}{1+q-q_kA_{s-1}};\\ B_s=\dfrac{\theta_{s,v}+q_k\left(\theta_{s-1,v}-2\theta_{s,v}+\theta_{s+1,v}\right)+q_kB_{s-1}}{1+q-q_kA_{s-1}}.\end{cases}$$

Отже, в схемі Кранка-Нікольсона прямий хід реалізується за формулами . Решта обчислень ідентичні таким в неявній схемі.

```
procedure StepKN(var Teta:CoefR);
var s:integer;
    A,B:CoefR;
    Zn:real;
begin
  t:=t+Tau;
  case Ngl of
    1:begin
        A[0]:=0; B[0]:=Tl(t)
      end;
    2:begin
        A[0]:=1; B[0]:=Hx/Lam*Ql(t)
      end;
```

```
    3:begin
        A[0]:=B0;
        B[0]:=B1*Tsl(t)
      end
  end;
  for s:=1 to m-1 do
    begin
      Zn:=1+q-qk*A[s-1];
      A[s]:=qk/Zn;
      B[s]:=((1-q)*Teta[s]+qk*(Teta[s-1]+Teta[s+1])+
             qk*B[s-1])/Zn
    end;
  case Ngr of
    1:Teta[m]:=Tr(t);
    2:Teta[m]:=(B[m-1]-Hx/Lam*Qr(t))/(1-A[m-1]);
    3:Teta[m]:=(B2*B[m-1]+B3*Tsr(t))/(1-B2*A[m-1])
  end;
  for s:=m-1 downto 0 do
    Teta[s]:=A[s]*Teta[s+1]+B[s]
end;
```

Комбінована схема, як і неявна, є стійкою при будь-яких *q*. Причина стійкості в тому, що похибки, які, можливо, виникають в процесі розв'язку на кожному черговому кроці за часом відносно рівномірно розподіляються між елементами масиву *Teta* (розв'язку) в процесі розв'язання системи рівнянь (методом прогонки). Тому лавиноподібного накопичення їх в окремих елементах масиву розв'язку не відбувається.

Комбінована схема дещо точніша від неявної за рахунок незначного збільшення обсягу обчислень в процесі виконання прямого ходу. Але, турбуючись про точність розв'язку ми не рекомендуємо зловживати її стійкістю.

Література

1. Аверіна Т.В., Кубрак Н.А. Динаміка елементів систем : Навч. посібник – К.: ІЗМН, 1998 – 224 с.

2. Карачун В.В., Кваско М.З., Кубрак Н.А. Прикладний аналіз і візуалізація характеристик динамічних систем : Навч. посібник – К.: ІЗМН, 1999 – 138 с.

3. Кваско М.З., Кубрак Н.А. Динамічні моделі типових теплообмінних апаратів : Навч. посібник – К.: ІЗМН, 1999 – 136 с.

4. Кубрак Н.А. Хвильові процеси в гнучких ланках автоматичних систем : Навч. посібник – К.: НМЦ ВО, 2000 – 160 с.

5. Калиткин Н.Н. Численные методы – М.: Наука, 1978 – 512 с.

6. Карачун В.В., Кубрак Н.А. Дротяні елементи приладів в акустичному середовищі – К.: “Корнійчук” , 2001 – 160 с.

7. Кошляков Н.С., Глинер Э.Б., Смирнов М.М. Основные дифференциальные уравнения математической физики – М.: Физмат, 1962 – 767 с.

8. Кубрак А.І. Ідентифікація динамічних характеристик елементів систем керування. Част. 1. Математичні методи : Навч. посібник – К.: ІСДО, 1995 – 208 с.

9. Меркин Д.Р. Введение в механику гибкой нити – М.: Наука, 1980 – 240 с.