TurboPascal - Prépa HEC Ipecom
Algorithmes classiques
notes de cours - vendredi 12 avril

1 Multiplication de matrices

On considère des matrices carrées. La taille et le type sont donnés par `const n=10; type matrice=array[1..n,1..n] of real;`.

On peut écrire la fonction suivante:

```
function m(a:matrice;b:matrice):matrice;
var i,j,k:integer;s:real
begin
  for i:=1 to n do
    for j:=1 to n do
      begin
        s:=0;
        for k:=1 to n
          do s:=s+a[i,k]*b[k,j]; m[i,j]=s;
      end;
end;
```

2 Dichotomie

La dichotomie consiste à chercher la racine d'une fonction , sur un intervalle pré-défini, en coupant celle-ci en deux jusqu'à l'obtention de la précision souhaitée. Ici, on suppose la fonction `function f(x:real):real;` définie, de même que la précision, par `const e=0.0001;`.

La procédure suivante recherche entre `a` et `b`, avec la précision `e`, une racine de notre fonction, et la renvoie dans la variable `x`:

```
procedure dicho(a,b:real; var x:real);
var x1,x2,x3,y1,y2,y3:real;
begin
  x1:=a; x2:=b;
  y1:=f(x1); y2:=f(x2);
  while abs(x1-x2)>e do
    begin
      x3:=(x1+x2)/2; y3:=f(x3);
      if y3*y2>0
        then
          begin x2:=x3; y2:=y3; end
        else begin x1:=x3; y1:=y3; end
    end;
  x:=(x1+x2)/2;
end;
```

3 Sécantes

La méthode des sécantes ne coupe pas l'intervalle $[a;b]$ en deux parties égales, mais le coupe là où le segment $[(a;f(a));(b;f(b))]$ croise l'axe des abscisses.

On a alors [seul le calcul de `x3` change]:

```
procedure sec(a,b:real; var x:real);
var x1,x2,x3,y1,y2,y3:real;
begin
  x1:=a; x2:=b;
  y1:=f(x1); y2:=f(x2);
  while abs(x1-x2)>e do
    begin
      x3:=x1-(x2-x1)*y1/(y2-y1)); y3:=f(x3);
      if y3*y2>0
        then begin x2:=x3; y2:=y3; end
        else begin x1:=x3; y1:=y3; end
    end;
  x:=(x1+x2)/2;
end;
```

4 Newton

Ici, au lieu de considérer le segment $[(a;f(a));(b;f(b))]$, on prend un seul point de départ a puis la droite tangente à notre graphe en a. On suppose donc la fonction `function f(x:real):real;` et sa dérivée `function ff(x:real):real;` définies, de même que la précision, par `const e=0.0001;`.

La procédure suivante recherche près de `a`, avec la précision `e`, une racine de notre fonction, et la renvoie dans la variable `x`:

```
procedure new(a:real; var x:real);
var x1,x2,y1,y2,yy1,yy2:real;
begin
  x1:=a; y1:=f(x1); yy1:=ff(x2);
  x2:=x1-y1/yy1; y2:=f(x2); yy2:=ff(x2);
  while abs(x1-x2)>e do
    begin
      x1:=x2; y1:=y2; yy1:=yy2;
      x2:=x1-y1/yy1; y2:=f(x2); yy2:=ff(x2);
    end;
  x:=(x1+x2)/2;
end;
```

5 Intégrales par rectangles

Pour approcher l'intégrale d'une fonction sur un intervalle, on découpe ce dernier en petits intervalle sur lesquells on suppose la fonction constante.

Après avoir défini la fonction `function f(x:real):real;` et le nombre d'intervalles `const n=100;`, on peut utiliser la procédure suivante, qui renvoie dans `i` l'intégrale entre `a` et `b`:

```
procedure int_reg(a,b:real; var i:real);
var k,h,x,y:real;
begin
  i:=0; h:=(b-a)/n; x:=a;
  for k:=1 to n do
    begin
      x:=x+h; y:=f(x); i:=i+h*y;
    end;
end;
```

6 Intégrales par trapèze

On ne suppose plus ici la fonction constante, mais linéaire.
La procédure suivante dait donc une somme de trapèzes:

```
procedure int_reg(a,b:real; var i:real);
var k,h,x1,x2,y1,y2:real;
begin
  i:=0; h:=(b-a)/n; x2:=a; y2:=f(x2);
  for k:=1 to n do
    begin
      x1:=x2; y1:=y2;
      x2:=x2+h; y2:=f(x2);
      i:=i+h*(y1+y2)/2;
    end;
end;
```

7 Suites par récurrence

Si la suite u_n est définie par $u_n = f(u_{n-1})$, avec u_0 donné, on affiche u_i par [avec `u0` et `f` donnés] par cette procédure:

```
procedure aff1(i:int);
var k,u:real;
begin
  u:=u0;
  for k:=1 to i do u:=f(u);
  writeln(u);
end;
```

Si la suite u_n est définie par $u_n = f(u_{n-1}, u_{n-2})$, avec u_0 et u_1 donnés, on affiche u_i par [avec `u0`, `u1` et `f` donnés]:

```
procedure aff2(i:int);
var k,u,v,w:real;
```

```
begin
  w:=u0; v:=u1;
  for k:=2 to i do
    begin u:=f(v,w); w:=v; v:=u; end;
  writeln(u);
end;
```

8 Polynômes

On cherche à obtenir le quotient et le reste de la division du polynôme P par $(x-a)$, c'est-à dire Q et $P(a)$, tels que $P=(x-a)Q+P(a)$.

L'algorithme de Horner indique que $q_0=p_0$, $q_i=p_i+a.q_{i-1}$, $P(a)=p_n+a.q_{n-1}$, où les p_i et q_i sont les coefficients de P et Q [cela permet également de calculer $P(a)$ sans calculs de puissances, donc rapidement]. Ici, on suppose le degré donné par `const n=10;`. Les coefficients seront stockés dans un tableau `type poly=array[0..n] of real;`.

La procédure ci-après construit le polynôme quotient Q dans `Q`, et la valeur $P(a)$ dans `x`:

```
procedure horner(P:poly; a:real; var Q:poly; var x:real);
var i;
begin
  Q[0]:=P[0];
  for i:=1 to n
    do Q[i]:=P[i]+a*Q[i-1];
  x:=Q[n]; Q[n]:=0;
end;
```

REMARQUE: On peut remarquer que

$$P(x)=p_0+p_1x+p_2x^2+p_3x^3+\ldots+p_{n-1}x^{n-1}+p_nx^n$$

$$=p_0+x(p_1+x(p_2+x(p_3+x(\ldots(p_{n_1}+x(p_n))\ldots))))$$

d'où le calcul de $P(a)$.