# TP3 - M43 Analyse numérique

Gloria Faccanoni

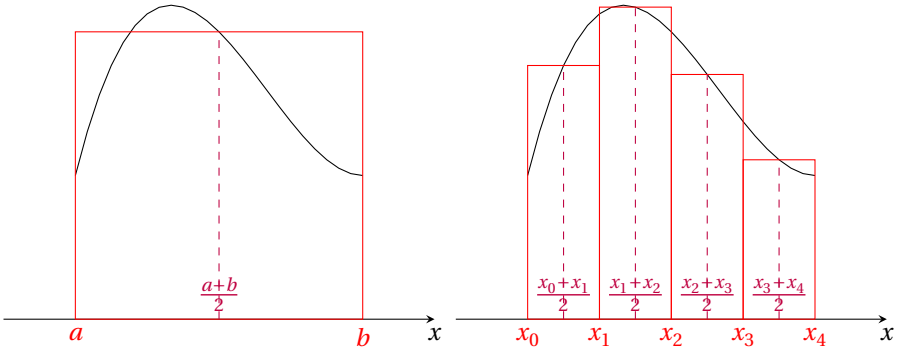
On s'intéresse dans ce TP aux diverses méthodes vues en cours et/ou en TD pour le calcul approché de l'intégration numérique d'une fonction réelle de variable réelle sur un intervalle et pour le calcul approché de la solution d'une EDO.

## 1 Exercice

On rappelle que si $n$ est un entier

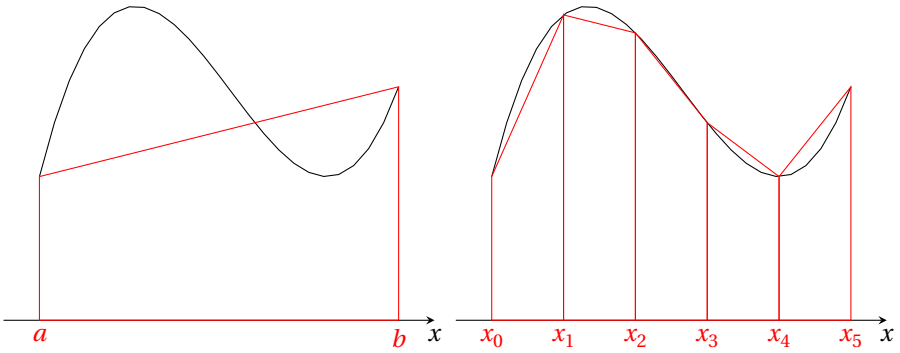
1. la formule du *rectangle* ou du *point milieu* à $n$ sous-intervalles pour calculer l'intégrale d'une fonction $f$ sur l'intervalle $[a,b]$ est

$$\int_a^b f(t)\,dt \simeq \mathscr{R}_n := h\sum_{i=0}^{n-1} f\left(a+\frac{2i+1}{2}h\right) \qquad \text{avec } h=\frac{b-a}{n};$$




2. la formule des *trapèzes* à $n$ points pour calculer l'intégrale d'une fonction $f$ sur l'intervalle $[a,b]$ est

$$\int_a^b f(t)\,dt \simeq \mathscr{S}_n := h\left(\frac{1}{2}f(a)+\sum_{i=1}^{n-1} f(a+ih)+\frac{1}{2}f(b)\right) \qquad \text{avec } h=\frac{b-a}{n};$$




3. la formule de *Cavalieri-Simpson* à $n$ points pour calculer l'intégrale d'une fonction $f$ sur l'intervalle $[a,b]$ est

$$\int_a^b f(t)\,dt \simeq \mathscr{S}_n := \frac{h}{6}\left(f(a)+2\sum_{r=1}^{n-1} f(a+rh)+4\sum_{s=0}^{n-1} f\left(a+\frac{2s+1}{2}h\right)+f(b)\right) \qquad \text{avec } h=\frac{b-a}{n};.$$

On sait que si $f$ est de classe $\mathscr{C}^3([a,b])$, on a

$$\lim_{n\to+\infty}\frac{1}{n}\left|\int_a^b f(t)\,dt - I_n\right| = 0, \qquad \text{pour } I_n=\mathscr{R}_n,\mathscr{T}_n,\mathscr{S}_n.$$

L'observer numériquement pour les fonctions suivantes :

$$f\colon x\mapsto \sin(x) \quad \text{sur } [0,\pi]; \qquad g\colon x\mapsto \frac{1}{1+x^2} \quad \text{sur } [0,1];$$
$$h\colon x\mapsto e^x \quad \text{sur } [-1,1]; \qquad w\colon x\mapsto xe^{-x} \quad \text{sur } [0,2\pi];$$

avec $n = 1,2,4,8,16,32,64,128$.

## 2 Exercice

Soit $I = [t_0, t_0+T]$ un intervalle fermé borné de $\mathbb{R}$. On se donne une fonction $f$ définie et continue sur $I\times\mathbb{R}$ à valeurs dans $\mathbb{R}$. Pour $\eta\in\mathbb{R}$, condition initiale, on cherche une fonction $y$ définie et dérivable sur $I$ à valeurs dans $\mathbb{R}$ telle que

$$\begin{cases} y'(t) = f(t,y(t)), & \forall t\in I,\\ y(t_0)=\eta. \end{cases} \tag{1}$$

Il s'agit d'un problème de Cauchy avec une EDO du premier ordre. On sait que s'il existe un réel $L$ tel que

$$\forall t\in I,\ \forall y,z\in\mathbb{R},\ |f(t,y)-f(t,z)|\le L|y-z|, \quad \text{(Condition de Lipschitz)}$$

alors le problème (1) admet une solution unique.

Rappelons ici trois schémas numériques : la méthode d'Euler explicite, la méthode de Heun (ou schéma de Runge-Kutta d'ordre 2) et la méthode de Runge-Kutta classique, d'ordre 4.

Soit $h = t_{i+1}-t_i$ ; alors on a

**Euler explicite**

$$\begin{cases} y_0=\eta,\\ y_{i+1} = y_i + hf(t_i,y_i), & i=0,\dots,n-1; \end{cases} \tag{2}$$

**Heun**

$$\begin{cases} y_0=\eta,\\ y_{i+1} = y_i + \frac{h}{2}\left(f(t_i,y_i)+f(t_{i+1},y_i+hf(t_i,y_i))\right), & i=0,\dots,n-1; \end{cases} \tag{3}$$

**RK4**

$$\begin{cases} y_0=\eta,\\ y_{i+1} = y_i + \frac{h}{6}\left(f(t_i,y_{i,1})+2f(t_i+h/2,y_{i,2})+2f(t_i+h/2,y_{i,3})+f(t_{i+1},y_{i,4})\right), & i=0,\dots,n-1; \end{cases} \tag{4}$$

avec

$$\begin{aligned} y_{i,1} &= y_i\\ y_{i,2} &= y_i + \frac{h}{2}f(t_i,y_{i,1})\\ y_{i,3} &= y_i + \frac{h}{2}f(t_i+h/2,y_{i,2})\\ y_{i,4} &= y_i + hf(t_i+h/2,y_{i,3}) \end{aligned}$$

Programmer ces méthodes comme des fonctions informatiques.

Pour chaque problème de Cauchy ci-dessous calculer la solution exacte et les solutions approchées obtenues par les méthodes précédentes et afficher les courbes ainsi obtenues sur un même plan (par exemple avec gnuplot) :

$$\begin{cases} y'(t) = -y(t),\\ y(0)=1, \end{cases} \tag{5}$$

$$\begin{cases} y'(t) = \frac{y(t)+x^2-2}{x+1},\\ y(0)=1, \end{cases} \tag{6}$$

$$\begin{cases} y'(t) = \cos^2(y(t)),\\ y(0)=0, \end{cases} \tag{7}$$

$$\begin{cases} y'(t) = x+y(t),\\ y(0)=1, \end{cases} \tag{8}$$

$$\begin{cases} y'(t) = xy(t),\\ y(0)=1. \end{cases} \tag{9}$$

Pour les calculs approchés on utiliera $h = 0.1$ et $n = 100$.

☎ 0033 (0)4 94 14 23 81
✉ gloria.faccanoni@univ-tln.fr
ℹ http://faccanoni.univ-tln.fr



IMATH Bâtiment U-318
Université du Sud Toulon-Var
Avenue de l'université
83957 LA GARDE - FRANCE

## 3 Code

integration.c

```c
/*
 * File:   integration.c
 * Author: Gloria Faccanoni
 *
 * Created on 11 mars 2010, 14:33
 */

#include <stdio.h>
#include <stdlib.h>
#include <math.h>

/*
 * gcc -o integration -lm integration.c && ./integration
 */
int main ( int argc, char** argv ) {

    double a, b ;
    int n, exo ;

    n = 30 ; //128;
    exo = 6 ;


    // Fonctions pour les integrations

    double rectangle ( double (*f )(double), double a, double b, int nb ) {
        double h, result ;
        int i ;
        result = 0 ;
        h = ( b - a ) / nb ;
        for ( i = 0 ; i < nb ; i++ )
            result += f ( a + ( 2.0 * i + 1 ) / 2.0 * h ) ;
        result *= h ;
        return result ;
    }

    double trapeze ( double (*f )(double), double a, double b, int nb ) {
        double h, result ;
        int i ;
        h = ( b - a ) / nb ;
        result = 0.0 ;
        for ( i = 1 ; i < nb ; i++ )
            result += f ( a + i * h ) ;
        result = ( f ( a ) + f ( b ) + result * 2.0 ) * h / 2.0 ;
        return result ;
    }

    double Simpson ( double (*f )(double), double a, double b, int nb ) {
        double h, result, s, r ;
        int i ;
        h = ( b - a ) / nb ;
        s = 0.0 ;
        r = 0.0 ;
        for ( i = 1 ; i < nb ; i++ )
            r += f ( a + i * h ) ;
        for ( i = 0 ; i < nb ; i++ )
            s += f ( a + ( 2.0 * i + 1.0 ) * h / 2.0 ) ;
        result = ( f ( a ) + f ( b ) + 2.0 * r + 4.0 * s ) * h / 6.0 ;
        return result ;
    }


```

```
63      // comparaison sur une fonction donnee
64
65      switch ( exo ) {
66          case 1:
67              a = 0.0 ;
68              b = M_PI ;
69              break ;
70          case 2:
71              a = 0.0 ;
72              b = 1.0 ;
73              break ;
74          case 3:
75              a = -1.0 ;
76              b = 1.0 ;
77              break ;
78          case 4:
79              a = 0.0 ;
80              b = M_PI + M_PI ;
81              break ;
82          case 5:
83              a = 0.0 ;
84              b = 1.0 ;
85              break ;
86          case 6:
87              a = 1.0 ;
88              b = 2.0 ;
89              break ;
90          default:
91              return 0 ;
92      }
93
94      double f ( double x ) {
95          switch ( exo ) {
96              case 1:
97                  return sin ( x ) ;
98                  break ;
99              case 2:
100                 return 1.0 / ( 1.0 + x * x ) ;
101                 break ;
102             case 3:
103                 return exp ( x ) ;
104                 break ;
105             case 4:
106                 return x * exp (-x ) * cos ( 2.0 * x ) ;
107                 break ;
108             case 5:
109                 return x * x ;
110                 break ;
111             case 6:
112                 return 1.0 / x ;
113                 break ;
114             default:
115                 return 0 ;
116         }// end swich
117     }// end f
118
119     double exacte ( double x ) {
120         switch ( exo ) {
121             case 1:
122                 return -cos ( x ) ;
123                 break ;
124             case 2:
125                 return atan ( x ) ;
126                 break ;
127             case 3:
128                 return exp ( x ) ;
```

```
129                 break ;
130             case 4:
131                 return (-x / 0.5e1 + 0.3e1 / 0.25e2 ) * exp (-x ) * cos ( 0.2e1 * x )
132                         - (-0.2e1 / 0.5e1 * x - 0.4e1 / 0.25e2 ) * exp (-x ) * sin ( 0.2e1 * x ) ;
133                 break ;
134             case 5:
135                 return x * x * x / 3.0 ;
136                 break ;
137             case 6:
138                 return log ( x ) ;
139                 break ;
140             default:
141                 return 0 ;
142         }// end swich
143     }// end f
144
145
146     printf ( "\nIntegration␣exacte␣entre␣%lf␣et␣%lf␣:", a, b ) ;
147     double integrale = exacte ( b ) - exacte ( a ) ;
148     printf ( "\nEXACTE\t\t␣%1.16lf\n\n", integrale ) ;
149
150     // Affiche les solutions approchees
151     printf ( "\nIntegration␣numerique␣avec␣%d␣points␣:", n ) ;
152     printf ( "\nRECTANGLE\t␣%1.16lf", rectangle ( f, a, b, n ) ) ;
153     printf ( "\nTRAPEZE\t\t␣%1.16lf", trapeze ( f, a, b, n ) ) ;
154     printf ( "\nSIMPSON\t\t␣%1.16lf\n\n", Simpson ( f, a, b, n ) ) ;
155
156     // Affiche les erreurs
157     printf ( "\nERREURS␣avec␣%d␣points␣entre␣%lf␣et␣%lf␣:", n, a, b ) ;
158     printf ( "\nRECTANGLE\t␣%1.16lf", fabs ( integrale - rectangle ( f, a, b, n ) ) ) ;
159     printf ( "\nTRAPEZE\t\t␣%1.16lf", fabs ( integrale - trapeze ( f, a, b, n ) ) ) ;
160     printf ( "\nSIMPSON\t\t␣%1.16lf\n\n", fabs ( integrale - Simpson ( f, a, b, n ) ) ) ;
161
162     return (EXIT_SUCCESS ) ;
163 }
```

edo.c

```c
/*
 * File:   edo.c
 * Author: Gloria Faccanoni
 *
 * Created on 17 mars 2010, 23:01
 */
#include <stdio.h>
#include <stdlib.h>
#include <math.h>

#define NB 100
#define NUMDEEE 5 // Choix du cas test

double *allouer_tableau ( int n ) {
    int i ;
    double *tab ;
    tab = (double *) malloc ( n * sizeof (double) ) ;
    if ( tab == NULL ) {
        printf ( "Allocation␣impossible␣!" ) ;
        exit ( 1 ) ;
    }
    return tab ;
}

/*
 * gcc -o edo -lm edo.c && ./edo && gnuplot
 * plot 'edo.dat' u 1:2 with line, 'edo.dat' u 1:3, 'edo.dat' u 1:5, 'edo.dat' u 1:7
 *
 */
int main ( int argc, char** argv ) {

    double *x, *y_euler, *y_heun, *y_rk4, h, ytam ;
    int i ;

    FILE *output ; /* internal filename */


    // Declaration des constantes de calcul
    h = 0.1 ; // pas pour le calcul

    //* Allocation de la memoire pour les tableaux de calcul
    x = allouer_tableau ( NB + 2 ) ;
    y_euler = allouer_tableau ( NB + 2 ) ;
    y_heun = allouer_tableau ( NB + 2 ) ;
    y_rk4 = allouer_tableau ( NB + 2 ) ;


    // C.I.
    switch ( NUMDEEE ) {
        case 1:
            x[0] = 0 ;
            y_euler[0] = 1 ;
            break ;
        case 2:
            x[0] = 0 ;
            y_euler[0] = 1 ;
            break ;
        case 3:
            x[0] = 0 ;
            y_euler[0] = 0 ;
            break ;
        case 4:
            x[0] = 0 ;
            y_euler[0] = 1 ;
```

```
            break ;
        case 5:
            x[0] = 0 ;
            y_euler[0] = 1 ;
            break ;
    }
    y_heun[0] = y_euler[0] ;
    y_rk4[0] = y_euler[0] ;

    // Terme source

    double f ( double x, double y ) {
        switch ( NUMDEEE ) {
            case 1: return -y ;
                break ;
            case 2: return (y + x * x - 2.0 ) / ( x + 1.0 ) ;
                break ;
            case 3: return pow ( cos ( y ), 2.0 ) ;
                break ;
            case 4: return x + y ;
                break ;
            case 5: return x * y ;
                break ;
        }
    }

    // Solution exacte

    double exacte ( double x ) {
        switch ( NUMDEEE ) {
            case 1: return exp (-x ) ;
                break ;
            case 2: return ( x + 1.0 / ( x + 1 ) - 2.0 * log ( x + 1 ) ) * ( x + 1 ) ;
                break ;
            case 3: return atan ( x ) ;
                break ;
            case 4: return 2. * exp ( x ) - x - 1.0 ;
                break ;
            case 5: return exp ( x * x / 2 ) ;
                break ;
        }
    }


    // Methode d'Euler

    double euler ( double x[], double y[], double h ) {
        int i ;
        for ( i = 0 ; i <= NB ; i++ ) {
            x[i + 1] = x[i] + h ;
            y[i + 1] = y[i] + h * f ( x[i], y[i] ) ;
        }
    }


    // Methode de Heun

    double heun ( double x[], double y[], double h ) {
        double k1, k2 ;
        int i ;
        for ( i = 0 ; i <= NB ; i++ ) {
            k1 = h * f ( x[i], y[i] ) ;
            k2 = h * f ( x[i] + h, y[i] + k1 ) ;
            y[i + 1] = y[i]+( k1 + k2 ) / 2.0 ;
        }
    }
```

```
131
132
133     // Methode de Runge Kutta 4
134
135     double runge4 ( double x[], double y[], double h ) {
136         double hh = h / 2.0, k1, k2, k3, k4 ;
137         int i ;
138         for ( i = 0 ; i <= NB ; i++ ) {
139             k1 = h * f ( x[i], y[i] ) ;
140             k2 = h * f ( x[i] + hh, y[i] + k1 / 2.0 ) ;
141             k3 = h * f ( x[i] + hh, y[i] + k2 / 2.0 ) ;
142             k4 = h * f ( x[i] + h, y[i] + k3 ) ;
143             y[i + 1] = y[i]+( k1 + 2 * k2 + 2 * k3 + k4 ) / 6.0 ;
144         }
145     }
146
147
148     // Coeur du programme
149     euler ( x, y_euler, h ) ;
150     heun ( x, y_heun, h ) ;
151     runge4 ( x, y_rk4, h ) ;
152
153     // Sauvegarde des resultats dans un fichier texte pour tracer par GnuPlot
154     // Ce fichier se nomme edo.dat. Il est cree dans le repertoire courant
155     // (celui dans lequel est lance le programme)
156     // S'il n'existe pas, il est cree. S'il existe, son contenu est ecrase.
157     output = fopen ( "edo.dat", "w+" ) ;
158     fprintf ( output,
159               "\n#␣x\t\ty_exacte\ty_euler\t\terreur\t\ty_heun\t\terreur\t\ty_runge\t\terreur\n"
160     for ( i = 0 ; i <= NB ; i++ ) {
161         ytam = exacte ( i * h ) ;
162         fprintf ( output, "%lf\t%lf\t%lf\t%lf\t%lf\t%lf\t%lf\t%lf\n",
163                   x[i], ytam,
164                   y_euler[i], ytam - y_euler[i],
165                   y_heun[i], ytam - y_heun[i],
166                   y_rk4[i], ytam - y_rk4[i] ) ;
167     }
168     fclose ( output ) ;
169
170     free ( x ) ;
171     free ( y_euler ) ;
172     free ( y_heun ) ;
173     free ( y_rk4 ) ;
174
175     //output=fopen("plot.gnu", "w+");
176     //fprintf(output,"plot 'edo.dat' u 1:2 with line, 'edo.dat' u 1:3, 'edo.dat' u 1:5, 'edo.da
177     //fclose(output);
178     //system("gnuplot plot.gnu -") ;
179
180     // AFFICHAGE GRAPHQUE:
181     output = popen ( "gnuplot", "w" ) ; // execution de la commande gnuplot
182     fprintf ( output, "␣plot␣\"edo.dat\"␣␣u␣1:2␣with␣line␣title␣'exact',␣'edo.dat'␣u␣1:3␣title␣
183     fflush ( output ) ;
184     sleep ( 10 ) ; // terminer l'envoi de commande
185     pclose ( output ) ; // fermer gnuplot
186
187     return 0 ;
188 }
189
190
191 /*
192 ode:=diff(y(x),x)=(y(x) + x * x - 2.0 ) / ( x + 1.0 );
193 dsolve({%,y(0)=1});
194 CodeGeneration[C](rhs(%));
195  */
```