Équations différentielles ordinaires

Méthode d'Euler-Cauchy
Méthode De Heun
Méthode de Runge-Kutta

© Pierre Lantagne
Enseignant retraité du Collège de Maisonneuve

La première version de ce document est parue en février 2006.

Nous connaissons plusieurs méthodes pour la résolution analytique de certaines équations différentielles du premier ordre et du premier degré. Par exemples, pour les

– équations différentielles à variables séparables

– équations différentielles homogènes

– équations différentielles linéaires

– équations différentielles de Bernouilli

– équations différentielles exactes

– équations différentielles non exactes

Mais, il y a plusieurs équations différentielles du premier ordre qui ne possèdent pas de solutions analytiques. Pour de telles équations, il existe plusieurs méthodes d'approximation numériques :

– méthode d'Euler-Cauchy

– méthode de Heun (ou méthode améliorée d'Euler-Cauchy)

– méthode de Runge-Kutta

Dans un premier temps, nous alllons éprouver la méthode d'Euler-Cauchy avec la solution analytique de l'équation différentielle du premier ordre $\frac{dy}{dx} = -2x - y$.

Bonne lecture à tous !

* Ce document Maple est exécutable avec la version 2020.2

Initialisation

```
> restart;
```

```
> with(plots,display,implicitplot,pointplot,setoptions):
  setoptions(axesfont=[times,roman,12],labelfont=[times,roman,12],titlefont=
  [times,italic,15]);
```

```
> with(DEtools,DEplot,dfieldplot):
```

Champ d'éléments de contact

Une équation différentielle du premier ordre est une équation de la forme $\frac{dy}{dx} = f(x, y)$. En se rappelant

l'interprétation graphique de $\frac{dy}{dx}$, f(x,y) est donc une formule donnant la pente de la tangente à la courbe solution passant par le point (x, y). Ainsi, l'équation différentielle $\frac{dy}{dx} = -2x - y$ peut être visualisée par un graphique appelé *champ d'éléments de contact.* Chaque élément de ce champ est un petit segment de droite centré au point (x_0, y_0) d'orientation $f'(x_0, y_0)$.

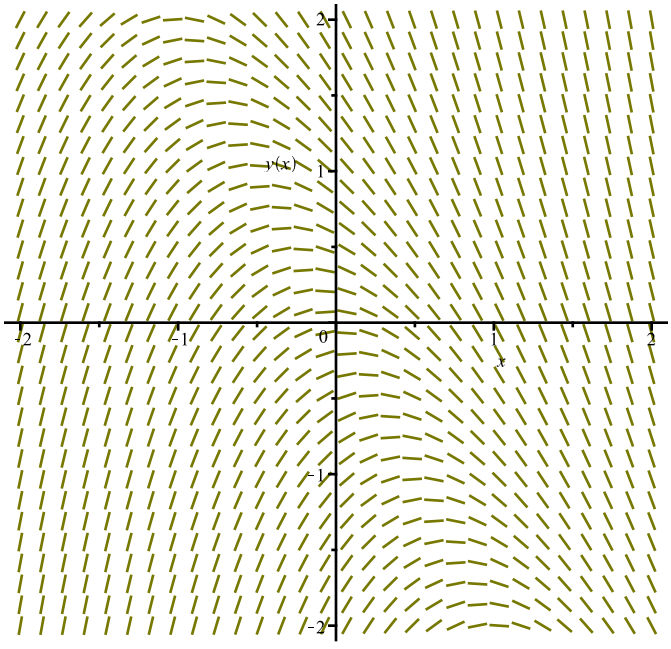
Ainsi, pour l'équation différentielle $\frac{dy}{dx} = -2x - y$, au point $(-2, 3)$, la pente de la tangente à la courbe solution y passant par ce point est $y' = 2 \cdot 2 - 3 = 1$. Au point $(6, 2)$ la pente de la tangente à la courbe solution est -14. Dans le premier cas, on illustre le résultat par un petit segment de droite centré en $(-2, 3)$ de pente 1 et, dans le second cas, par un autre segment de droite centré en $(6, 2)$ de pente -14. En répétant ce précessus pour un certain nombre de couples (x, y), on obtient ce qu'on appelle un *champ d'éléments de contact*, parfois traduit de l'anglais par champ de pentes.

La macro-commande `dfieldplot` de la bibliothèque `DEtools` permet le tracé d'un champ d'éléments de contact. La bibliothèque `DEtools` impose la formulation explicite de la variable dépendante. Dans le cas d'une fonction de deux variables x et y, si la variable y est désignée comme dépendante, on doit l'énoncer dans les requêtes avec la syntaxe $(y(x))$.

```
> ED:=diff(y(x),x)=-2*x-y(x);
```

$$ED := \frac{d}{dx} y(x) = -2x - y(x) \qquad \textbf{(2.1)}$$

```
> Champ:=dfieldplot(ED,[y(x)],x=-2..2,y=-2..2,arrows=line,dirgrid=[30,30],
  color="Niagara 16"):
  Champ;
```



On peut également représenter dans un champ d'éléments de contact, des courbes appelées **isoclines** (du grec *iso* qui signifie même et du latin *clinare* qui signifie pencher). Les isoclines sont des courbes le long desquelles les éléments de contact ont une direction (une inclinaison) donnée.

Superposer, dans le champ d'éléments de contact précédent, les isoclines de pente -1, 0 et 3.

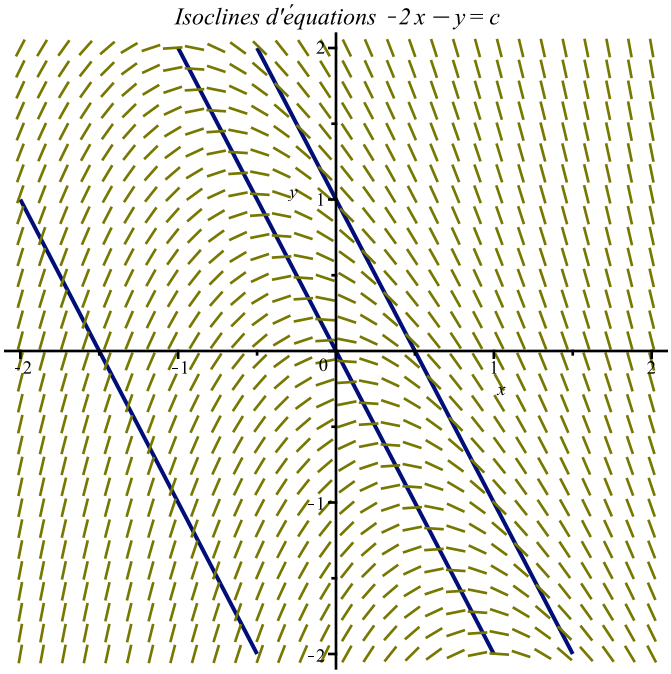
```
> f:=(x,y)->-2*x-y;
```

$$f := (x,y) \mapsto -2\cdot x - y \tag{2.2}$$

```
> Pentes:=[-1,0,3];
```

$$Pentes := [-1, 0, 3] \tag{2.3}$$

```
> for i from 1 to nops(Pentes) do
   Pente:=Pentes[i];
   isocline||i:=implicitplot(f(x,y)=Pente,x=-2..2,y=-2..2,color="Niagara 2"):
  od:
> display({isocline||(1..nops(Pentes)),Champ},title=typeset("Isoclines
  d'équations ", f(x,y)=c));
```



L'intérêt d'un champ d'éléments de contact apparaît clairement avec l'usage de l'ordinateur. Que l'équation différentielle possède ou non une solution analytique, ce type de tracé permet de visualiser l'ensemble des courbes solution de l'équation différentielle. En effet, le champ d'éléments de contact précédent permet de visualiser ceci:

– lorsque $x < -\frac{y}{2}$, chaque courbe solution est croissante

– lorsque $x > -\frac{y}{2}$, chaque courbe solution est décroissante

– lorsque $x = -\frac{y}{2}$, chaque courbe solution admet un donc maximum relatif

– lorsque $x \to \infty$, toutes les courbes solutions ont un comportement asymptotique

Puisque $y' = x - y$ est une équation différentielle du premier ordre et du premier degré, même s'il est facile

d'en obtenir directement la solution par un calcul à la main, résolvons quand même cette équation avec Maple.

```
> dsolve(ED);
```

$$y(x) = -2x + 2 + e^{-x}\_C1 \tag{2.4}$$

La solution générale de cette équation est donc

```
> Sol:=y=-2*x+2+C*exp(-x);
```

$$Sol := y = -2x + 2 + C e^{-x} \tag{2.5}$$

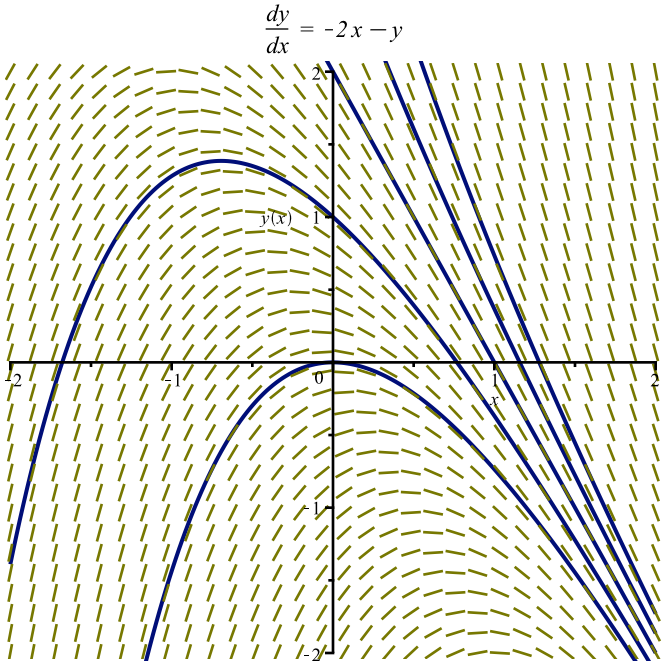
Superposons au champ d'éléments de contact précédent, les solutions particulières correspondant aux conditions initiales $y(0) = C$ pour C = −2, −1, 0, 1 et 2.

```
> Valeurs:=[-2,-1,0,1,2];
```

$$Valeurs := [-2, -1, 0, 1, 2] \tag{2.6}$$

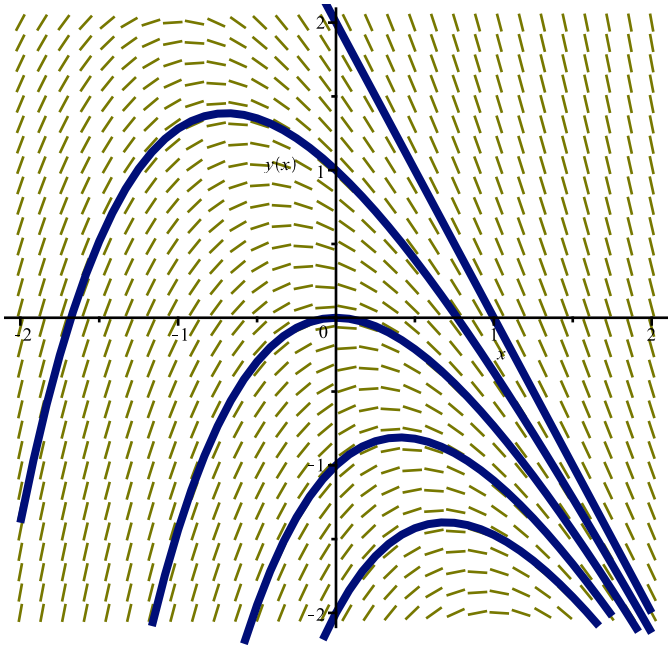
```
> Sol:=-2*x+2+C*exp(-x):
  for i from 1 to nops(Valeurs) do
   C:=Valeurs[i];
   f||i:=plot([x,Sol,x=-2..2],color="Niagara 2"):
  od:
  C:='C':
  i:='i':
```

```
> display([f||(1..nops(Valeurs)),Champ],view=[-2..2,-2..2],title=typeset(dy/dx
  = -2*x-y));
```



La macro-commande `DEplot` de la bibliothèque `DEtools` permet directement la superposition du tracé d'un champ d'éléments de contact avec ceux des solutions particulières.

```
> DEplot(ED,y(x),x=-2..2,[[y(0)=-2],[y(0)=-1],[y(0)=0],[y(0)=1],[y(0)=2]],y=
  -2..2,linecolor="Niagara 2",color="Niagara 16",arrows=line,dirgrid=[30,30]);
```



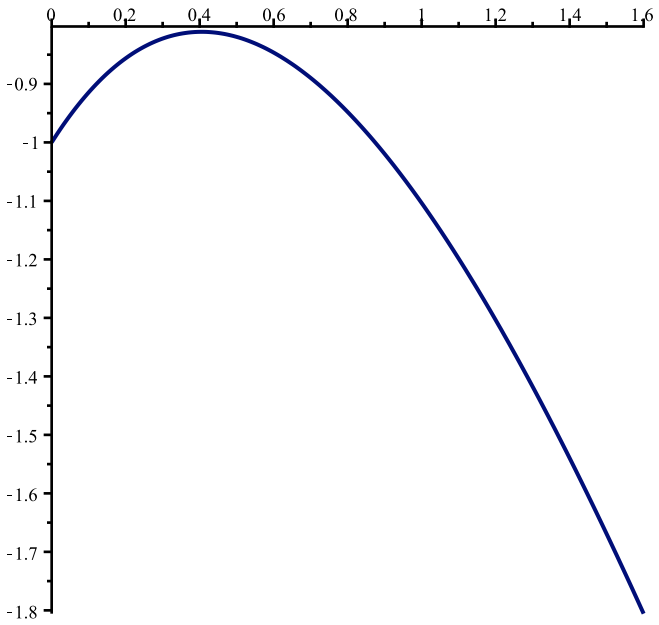
Méthode d'Euler-Cauchy

Mettons en évidence la solution particulière $y(0) = -1$ pour x $\in [0, 1.6]$.

```
> Sol_par:=dsolve({ED,y(0)=-1},y(x));
```

$$Sol\_par := y(x) = -2x + 2 - 3\,e^{-x} \qquad \textbf{(3.1)}$$

```
> plot([x,rhs(Sol_par),x=0..1.6],color="Niagara 2");
```



Ce tracé est exact car il a été obtenu à l'aide de la solution générale. Obtenons maintenant une approximation de ce tracé en joignant les points calculés avec la méthode d'Euler-Cauchy. En fait, il s'agit d'approximer la courbe solution de l'équation différentielle $\frac{dy}{dx} = y' = f(x, y)$ avec la condition initiale $y(x_0) = y_0$.

Cette méthode repose sur le processus itératif suivant. La condition initiale précise un premier point. Ce point est évidemment exact. Soit (x_0, y_0) ce premier point. En ce point, la pente de la tangente à la courbe solution est donnée par $f'(x_0, y_0)$. Alors, l'équation de la tangente en ce point est

$$y = y'(x_0)\ (x - x_0)\ + y(x_0)$$

c'est-à-dire

$$y = f'(x_0, y_0)\ (x - x_0)\ + y_0$$

L'expression $(x - x_0)$ sera vue comme un accroissement d'abscisse $h > 0$ qui sera « assez » petit. À cet accroissement, correspondra une valeur d'ordonnée jusqu'à la tangente $y_1 = f'(x_0, y_0)\ h + y_0$. On obtient ainsi un deuxième point $(x_1, y_1) = (x_0 + h, f'(x_0, y_0)\ h + y_0)$.

De même, un troisième point de coordonnées $(x_2, y_2) = (x_1 + h, f'(x_1, y_1)\ h + y_1)$ sera obtenu à l'aide de la direction de la tangente passant par le deuxième point f(x_1, y_1).

Et ainsi de suite: $(x_n, y_n) = (x_{n-1} + h, f'(x_{n-1}, y_{n-1})\ h + y_{n-1})$.

REMARQUE: La solution d'une équation différentielle est une fonction. La méthode d'Euler-Cauchy ne donne pas une fonction: elle permet seulement d'obtenir une séquence de couple (x_i, y_i) qu'il suffit de relier de manière appropriée pour approcher la courbe solution.

Appliquons maintenant cette approche pour obtenir une série de points. Ces points seront ensuite superposés dans un même graphique au tracé de la courbe solution. Nous pourrons alors faire une comparaison.

Automatisons le calcul d'un nombre n de couples (x, y). Rappelons-nous d'abord l'équation différentielle à résoudre.

```
> ED;
```

$$\frac{d}{dx}\ y(x) = -2x - y(x) \qquad \textbf{(3.2)}$$

La condition initiale impose le premier point.

```
> X[0]:=0;
  Y[0]:=-1;
  Points_Euler:=[X[0],Y[0]];
```

$$X_0 := 0$$

$$Y_0 := -1$$

$$Points\_Euler := [0, -1] \qquad \textbf{(3.3)}$$

Posons un accroissement $h = 0,1$.

```
> h:=0.1;
```

$$h := 0.1000000000 \qquad \textbf{(3.4)}$$

Répartissons uniformément le nombre N supplémentaires de points en calculant la largeur de l'intervalle d'approximation [0; 1,6] divisée par la valeur de l'accroissement h.

```
> N:=(1.6-0)/h;
```

$$N := 16.0000000000 \qquad \textbf{(3.5)}$$

Calculons ces 16 autres points avec une boucle.

```
> for n from 0 to N-1 do
    X[n+1]:=X[n]+h:
    Y[n+1]:=Y[n]+h*f(X[n],Y[n]):  # f(x,y) est la formule de calcul de la pente
    Tampon:=[X[n+1],Y[n+1]];
```

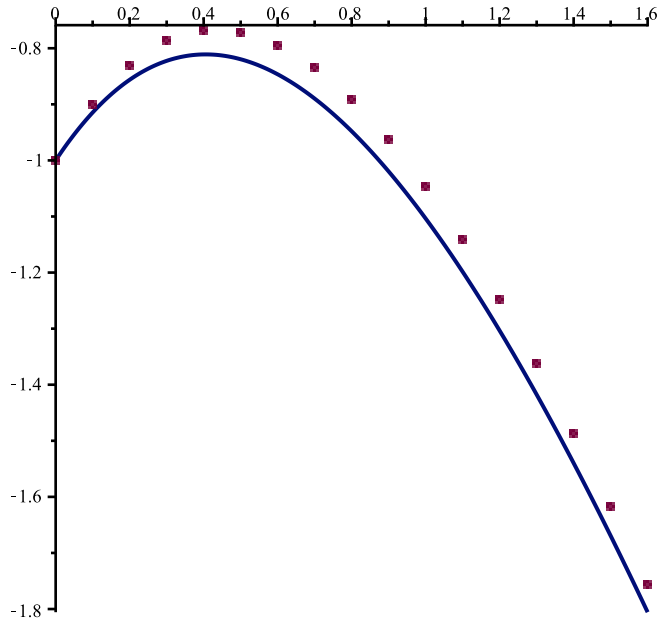
```
 Points_Euler:=Points_Euler,Tampon
od:
```

Superposons le tracé de ces points (sans les relier) avec celui de la courbe solution lorsque $y(0) = -1$.

```
> Sol:=dsolve({ED,y(0)=-1},y(x));
```

$$Sol := y(x) = -2x + 2 - 3e^{-x} \quad \textbf{(3.6)}$$

```
> Courbe_solution:=plot([x,rhs(Sol),x=0..1.6],color="Niagara 2"):
  Points_approx:=plot([Points_Euler],style=point,symbol=solidcircle,color=
  "Niagara 12"):
  display([Courbe_solution,Points_approx]);
```



Diminuons le valeur de l'accroissement pour une seconde approximation de la courbe solution.

```
> Points_Euler:=[X[0],Y[0]];
```

$$Points\_Euler := [0, -1] \quad \textbf{(3.7)}$$

```
> h:=0.05;
```

$$h := 0.0500000000 \quad \textbf{(3.8)}$$

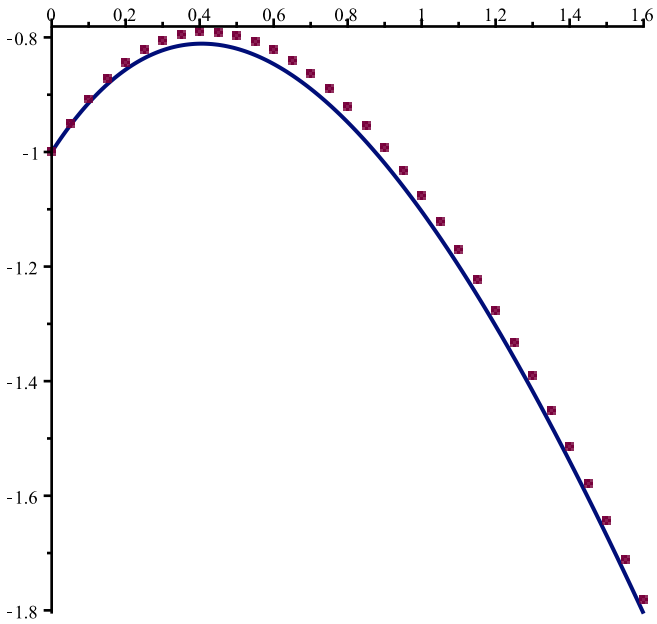
```
> N:=(1.6-0)/h;
  for n from 0 to N-1 do
   X[n+1]:=X[n]+h:
   Y[n+1]:=Y[n]+h*f(X[n],Y[n]):  # f(x,y) est la formule de calcul de la pente
   Tampon:=[X[n+1],Y[n+1]];
   Points_Euler:=Points_Euler,Tampon
  od:
```

$$N := 32.0000000000 \quad \textbf{(3.9)}$$

```
> Sol:=dsolve({ED,y(0)=-1},y(x));
```

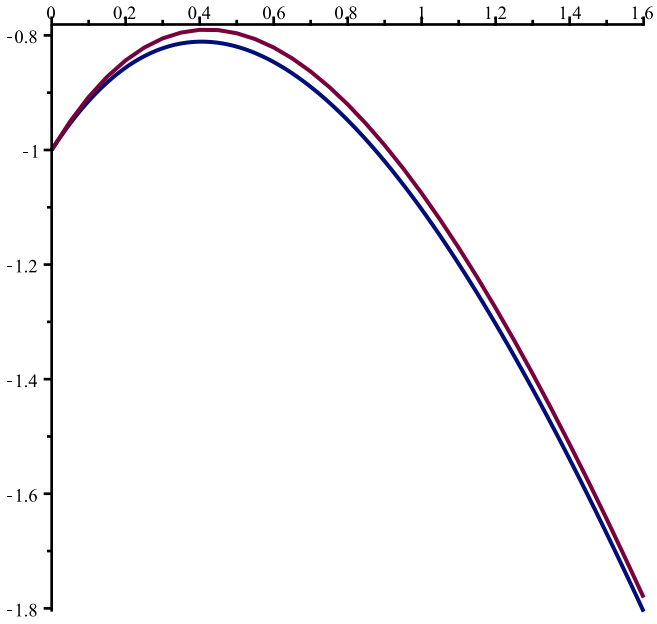
$$Sol := y(x) = -2x + 2 - 3e^{-x} \quad \textbf{(3.10)}$$

```
> Points_approx:=plot([Points_Euler],style=point,symbol=solidcircle,color=
  "Niagara 12"):
  display([Courbe_solution,Points_approx]);
```



Relions finalement ces points avec la macro-commande `pointplot` de la bibliothèque `plots`.

```
> Points_approx:=pointplot([Points_Euler],connect=true,color="Niagara 12"):
  display([Courbe_solution,Points_approx]);
```



Méthode d'Euler-Cauchy améliorée ou méthode de Heun

La méthode de Heun a comme caractéristique que l'ordonnée du prochain point est calculé non pas avec la pente de la tangente à la courbe solution du point courant mais plutôt avec la moyenne arithmétique des pentes des tangentes à la courbe solution du point courant et du point suivant temporaire calculé dans la direction donnée par la tangente au point suivant et ce, à partir de l'estimation de l'ordonnée du point courant...ouf!

Dans le cas de la méthode de Heun, nous avons:

$$x_{n+1} = x_n + h$$
$$u_{n+1} = y_n + h f'(x_n, y_n)$$
$$y_{n+1} = y_n + h \frac{f'(x_n, y_n) + f'(x_{x+1}, u_{n+1})}{2}$$

Pour bien automatiser le calcul des points de la courbe solution à estimer, automatisons dans une premiêre étape la méthode d'Euler-Cauchy avec la procédure ci-dessous. Dans une seonde étape, nous allons modifier cette procédure pour inclure un calcul intermédiaire pour pouvoir faire la moyenne.

```
> Euler:=proc(f::procedure,y0::realcons,Intervalle::range,h::realcons)
  global Points_Euler:
  local n,N,Tampon,X,Y;
  X[0]:=op(1,Intervalle); Y[0]:=y0;
  Points_Euler:=[X[0],Y[0]]:  # Point de la condition initiale
  N:=(op(2,Intervalle)-X[0])/h:
  for n from 0 to N-1 do
   X[n+1]:=X[n]+h:
   Y[n+1]:=Y[n]+h*f(X[n],Y[n]):  # f(x,y) est la formule de calcul de la pente
   Tampon:=[X[n+1],Y[n+1]];
   Points_Euler:=Points_Euler,Tampon
  od
  end:
```

Maintenant, voici l'algorithme de la méthode d'Euler-Cauchy améliorée, c'est-à-dire de la méthode de Heun.

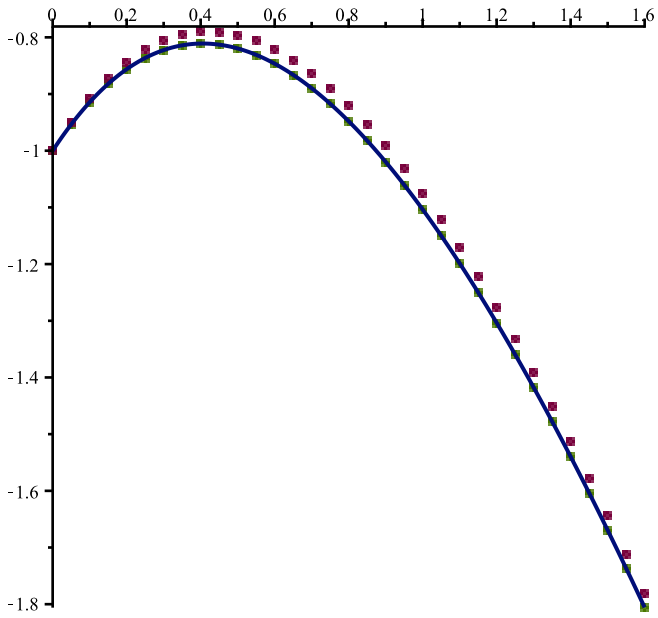
```
> Heun:=proc(f::procedure,y0::realcons,Intervalle::range,h::realcons)
  global Points_Heun;
  local n,N,Tampon,X,Y,U;
  X[0]:=op(1,Intervalle); Y[+0]:=y0;
  Points_Heun:=[X[0],Y[0]]:  # Point de la condition initiale
  N:=(op(2,Intervalle)-X[0])/h:
  for n from 0 to N-1 do
   X[n+1]:=X[n]+h:
   U[n+1]:=Y[n]+h*f(X[n],Y[n]):  # f(x,y) est la formule de calcul de la pente
   Y[n+1]:=Y[n]+h/2*(f(X[n],Y[n])+f(X[n+1],U[n+1])):
   Tampon:=[X[n+1],Y[n+1]];
   Points_Heun:=Points_Heun,Tampon
  od
  end:
```

Maintenant, comparons graphiquement la méthode de Heun avec celle d'Euler-Cauchy toujours avec l'équation différentielle du début $\frac{dy}{dx} = -2x - y$ avec comme condition initiale $y(0) = -1$.

```
> f:=(x,y)->-2*x-y;
```

$$f := (x, y) \mapsto -2 \cdot x - y \qquad \textbf{(4.1)}$$

```
> Euler_Approx:=plot([Euler(f,-1,0..1.6,0.05)],style=point,symbol=solidcircle,
  color="Niagara 12"):
  Heun_Approx:=plot([Heun(f,-1,0..1.6,0.05)],style=point,symbol=solidcircle,
  color="Niagara 3"):
  Champ:=dfieldplot(ED,[y(x)],x=0..1.6,y=-2..-0.5,
           arrows=line,dirgrid=[15,15],scaling=constrained,color=khaki):
  display([Heun_Approx,Euler_Approx,Courbe_solution]);
```



On constate effectivement que la méthode de Heun (points verts) améliore beaucoup l'estimation de la courbe solution obtenue avec la méthode d'Euler-Cauchy (points magenta).

Méthode de Runge-Kutta

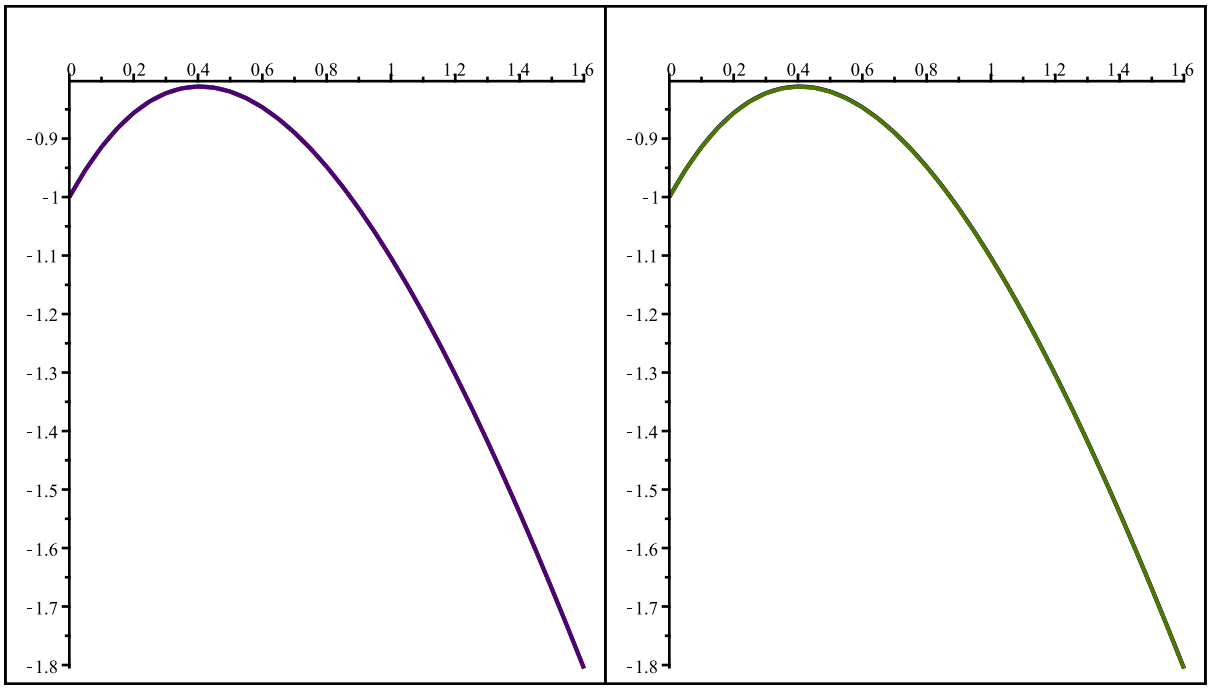
La méthode de Runge-Kutta du quatrième ordre améliore davantage la méthode de Euler-Cauchy que ne le fait la méthode de Heun mais les calculs sont encore beaucoup plus élaborés. Cette méthode repose aussi en quelque sorte sur un calcul de moyenne de pentes. Présentons, sans autres explications, la procédure automatisant l'algorithme de Runge-Kutta.

```
> Runge_Kutta:=proc(f::procedure,y0::realcons,Intervalle::range,h::realcons)
  global Points_Runge_Kutta;
  local n,N,p,q,r,s,Tampon,X,Y;
  X[0]:=op(1,Intervalle); Y[0]:=y0;
  Points_Runge_Kutta:=[X[0],Y[0]]:  # Point de la condition initiale
  N:=(op(2,Intervalle)-X[0])/h:
  for n from 0 to N-1 do
   X[n+1]:=X[n]+h:
   p[n]:=f(X[n],Y[n]):
   q[n]:=f(X[n]+h/2,Y[n]+h/2*p[n]):
   r[n]:=f(X[n]+h/2,Y[n]+h/2*q[n]):
   s[n]:=f(X[n]+h,Y[n]+h*r[n]):
   Y[n+1]:=Y[n]+h/6*(p[n]+2*q[n]+2*r[n]+s[n]):
   Tampon:=[X[n+1],Y[n+1]];
   Points_Runge_Kutta:=Points_Runge_Kutta,Tampon
  od
  end:
```

Pour l'équation différentielle $\frac{dy}{dx} = -2x - y$, la méthode de Heun montrait à la section précédente que les points calculés épousaient déjà, de manière remarquable, la courbe solution. Le degré de précision atteint avec cette méthode a pu être apprécié par une approche graphique à l'égard de la méthode d'Euler-Cauchy. Par contre, avec la méthode du Runge-Kutta, le degré de précision atteint à l'égard de la méthode de Heun est difficilement appréciable graphiquement (à l'écran). En effet, constatons que les tracés des courbes en reliant les points calculés par les méthodes de Heun et de Runge-Kutta et la courbe solution se superposent presque

parfaitement compte tenu de la résolution écran. Il est nécessaire de procéder en un agrandissement du graphique (avec la loupe) pour pouvoir « visualiser » l'augmentation de la précision.

```
> C1:=pointplot([Heun(f,-1,0..1.6,0.05)],connect=true,color="Niagara 3"):
  C2:=pointplot([Runge_Kutta(f,-1,0..1.6,0.05)],connect=true,color="Niagara
  15"):
  A1:=display([Courbe_solution,C1,C2]):
  A2:=display([Courbe_solution,C2,C1]):
  display(Matrix(1,2,[A1,A2]));
```



Seule une comparaison numérique peut permettre une comparaison claire de la précision atteinte par chaque méthode à l'égard de la courbe solution.

Soit la procédure suivante automatisant le calcul des coordonnées des points de la courbe solution sur la base de l'intervalle traité ainsi que la valeur commune *h* retenue dans les trois méthodes de calcul.

```
> Solution:=proc(ED::equation,y0::realcons,Intervalle::range,h::realcons)
  global Points_solution;
  local n,N,Sol,Tampon,X,Y,U;
  Sol:=unapply(rhs(dsolve({ED,y(0)=-1},y(x))),x);
  X[0]:=op(1,Intervalle); Y[0]:=y0;
  Points_solution:=[X[0],Y[0]]:  # Point de la condition initiale
  N:=(op(2,Intervalle)-X[0])/h:
  for n from 0 to N-1 do
   X[n+1]:=X[n]+h:
   Y[n+1]:=Sol(X[n+1]):
   Tampon:=[X[n+1],Y[n+1]];
   Points_solution:=Points_solution,Tampon
  od
```

```
end:
> Solution(ED,-1,0..1.6,0.05):
```

Obtenons maintenant un tableau donnant les écarts à la courbe solution pour chacune des méthodes d'approximation.

```
> printf("\n \n                    Écarts numériques à la courbe solution \n
                      y(x) = -2*x + 2 - 3*exp(-x)\n" );
  printf("\n       |  n  |   Écart Euler   |   Écart Heun    | Écart
  Runge_Kutta |\n");
  printf("       ================================================================
  ==\n");
   for k from 1 to nops([Points_Euler])
   do
   printf(`       | %2d  |  %+10.10f  |  %+10.10f  |   %+10.10f   | \n`,k,
          Points_solution[k,2]-Points_Euler[k,2],
          Points_solution[k,2]-Points_Heun[k,2],
          Points_solution[k,2]-Points_Runge_Kutta[k,2]) od:
```

Écarts numériques à la courbe solution
y(x) = -2*x + 2 - 3*exp(-x)

n	Écart Euler	Écart Heun	Écart Runge_Kutta
1	+0.0000000000	+0.0000000000	+0.0000000000
2	-0.0036882740	+0.0000617260	+0.0000000072
3	-0.0070122540	+0.0001174335	+0.0000000148
4	-0.0099989290	+0.0001675612	+0.0000000213
5	-0.0126735090	+0.0002125210	+0.0000000269
6	-0.0150595365	+0.0002526980	+0.0000000319
7	-0.0171789901	+0.0002884515	+0.0000000362
8	-0.0190523807	+0.0003201177	+0.0000000403
9	-0.0206988441	+0.0003480098	+0.0000000438
10	-0.0221362258	+0.0003724206	+0.0000000466
11	-0.0233811613	+0.0003936239	+0.0000000495
12	-0.0244491542	+0.0004118738	+0.0000000518
13	-0.0253546450	+0.0004274082	+0.0000000539
14	-0.0261110802	+0.0004404483	+0.0000000555
15	-0.0267309737	+0.0004512006	+0.0000000570
16	-0.0272259676	+0.0004598569	+0.0000000579
17	-0.0276068861	+0.0004665966	+0.0000000589
18	-0.0278837904	+0.0004715850	+0.0000000590
19	-0.0280660237	+0.0004749780	+0.0000000600
20	-0.0281622620	+0.0004769190	+0.0000000600
21	-0.0281805560	+0.0004775410	+0.0000000590
22	-0.0281283670	+0.0004769700	+0.0000000600
23	-0.0280126150	+0.0004753180	+0.0000000600
24	-0.0278397040	+0.0004726930	+0.0000000600
25	-0.0276155620	+0.0004691940	+0.0000000590
26	-0.0273456710	+0.0004649120	+0.0000000580
27	-0.0270350950	+0.0004599330	+0.0000000580
28	-0.0266885120	+0.0004543340	+0.0000000570
29	-0.0263102360	+0.0004481870	+0.0000000560
30	-0.0259042410	+0.0004415600	+0.0000000560
31	-0.0254741880	+0.0004345130	+0.0000000550
32	-0.0250234440	+0.0004271040	+0.0000000540
33	-0.0245551010	+0.0004193840	+0.0000000530

Cet exemple montre bien que la méthode de Runge Kutta permet d'être plus ajusté à la courbe solution.