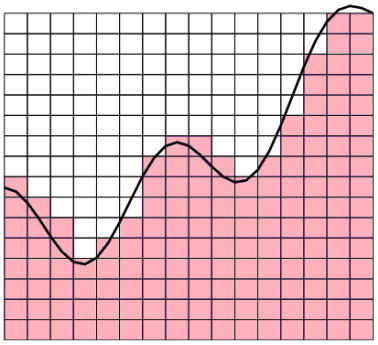
# 1 L'integrazione numerica

Consideriamo una funzione $f : [a, b] \to \mathbb{R}$ *continua* ed *integrabile* su $[a, b]$, di cui siamo interessati a calcolare l'**integrale definito**

$$I(f) = \int_a^b f(x)\, dx. \tag{1}$$

Molto spesso, questo calcolo presenta diverse difficolta: a volte $I(f)$ può non essere espresso tramite funzioni elementari, oppure i calcoli possono essere molto complessi da svolgere analiticamente. Sorge il bisogno, dunque, di considerare delle **formule di quadrature** per l'**integrazione numerica**.

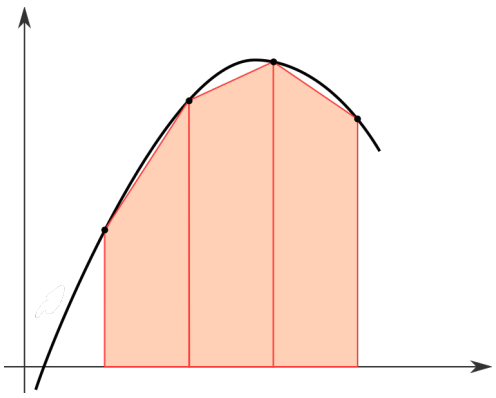
Il termine "*quadratura*" è un termine storico che si riferisce al processo di calcolo dell'area. Esso ricorda la tecnica elementare per determinare approssimativamente l'aera sottesa al grafico di una funzione: contare i quadratini sotto la curva, e moltiplicare questo numero per l'aerea di un singolo quadratino.



La strategia usata in analisi numerica è quella di sostituire ad $f$ una sua approssimazione $f_n$ con $n \in \mathbb{N}$, e calcolare un **integrale approssimato**

$$I_n(f) = \int_a^b f_n(x)\, dx, \tag{2}$$

che approssima $I(f)$ in maniera arbitariamente accurata.



Una misura importante dell'accuratezza di una formula di quadratura è il **grado di esattezza** di $I_n$, che è definito come il più grande intero $r \geq 0$ tale dhe

$$I_n(p_r) = I(p_r) \quad \forall p_r \in \mathbb{P}_r. \tag{3}$$

Tipicamente, l'approssimazione $f_n$ è ottenuta tramite tecniche di interpolazione polinomiale, come visto nelle Lezioni 16-18. Sia $f_n = \Pi_n f \in \mathbb{P}_n$ il polinomio interpolatore di Lagrange di $f$ rispetto ai $n+1$ nodi *distinti* $x_i \in [a,b]$, $i = 0, \ldots, n$. Sostituendo troviamo la formula di quadratura corrispondente

$$I_n(f) = \int_a^b \Pi_n f(x)\, dx = \int_a^b \left( \sum_{i=0}^{n} f(x_i)\ell_i(x) \right) dx = \sum_{i=0}^{n} f(x_i) \underbrace{\int_a^b \ell_i(x)\, dx}_{:=w_i} = \sum_{i=0}^{n} w_i f(x_i), \tag{4}$$

dove $x_i$ e $w_i$ vengono chiamati rispettivamente i **nodi** e i **pesi** della formula $I_n$. Formule di questo tipo si dicono **formule di quadratura interpolatorie**.

## 1.1 Le istruzioni `quad`, `quadl` e `integral`

⋆ L'istruzione `q=quad(fun,a,b)` di MATLAB approssima l'integrale di una funzione data `fun`, da $a$ a $b$ con una tolleranza d'errore di default di $1e-6$ utilizzando la **formula adattativa ricorsiva di Simpson**. La funzione `fun` dev'essere un function handle e i limiti $a$ e $b$ devono essere finiti. La funzione `y=fun(x)` ammette array come vettori d'input e ritorna array come vettori d'output, avendo valutata l'integranda in ogni elemento $x$.

Ad esempio, per calcolare l'integrale

$$\int_0^2 \frac{1}{x^3 - 2x - 5} \tag{5}$$

una possibilità è di creare un M-file di tipo function e poi calcolare l'integrale secondo la sintassi

```
function y = myfun(x)
  y = 1./(x.^3-2*x-5);
end
Q = quad(@myfun,0,2)
```

Alternativamente, si può direttamente usare un function handle

```
F = @(x) 1./(x.^3-2*x-5);
Q = quad(F,0,2)
```

⋆ Equivalentemente, l'istruzione `q=quadl(fun,a,b)` di MATLAB approssima l'integrale di una funzione data `fun`, da $a$ a $b$ con una tolleranza d'errore di default di $1e-6$ utilizzando la formula **adattativa ricorsiva di Gauss-Lobatto**.

Attualmente, MATLAB avvisa per entrambi i comandi: `quad/quadl will be removed in a future release. Use INTEGRAL instead.`

⋆ L'istruzione `q=integral(fun,xmin,xmax)` di MATLAB calcola numericamente l'integrale di una funzione data `fun` da `xmin` a `xmax` utilizzando **quadratura adattativa globale** e tolleranze d'errore di default.

Questo comando, permette di calcolare **integrali impropri**. Ad esempio, per calcolare l'integrale

$$\int_0^{+\infty} e^{-x^2}(lnx)^2\,dx \tag{6}$$

basta eseguire le istruzioni

```
fun = @(x) exp(-x.^2).*log(x).^2;
q = integral(fun,0,Inf)
```

Per calcolare l'integrale della **funzione parametrica**

$$f(x) = \frac{1}{x^3 - 2x - c} \tag{7}$$

specificando a piacere gli estremi d'integrazion ed il paramentro $c$, basta eseguire le istruzioni

```
fun = @(x,c) 1./(x.^3-2*x-c);
q = integral(@(x)fun(x,5),0,2)
```

L'istruzione `integral` permette inoltre di calcolare integrali di funzioni che presentono **singolarità** agli estremi. Ad esempio, per calcolare l'integrale

$$\int_0^1 ln(x)\,dx, \tag{8}$$

si eseguono le istruzioni

```
fun = @(x)log(x);
format long
q1 = integral(fun,0,1)
%valutiamo nuovamente l'integrale con piu' accuratezza
q2 = integral(fun,0,1,'RelTol',0,'AbsTol',1e-12)
```

Per più informazioni sulle istruzioni MATLAB inerenti alla quadratura e all'integrazione numerica si faccia riferimento alle pagine di referenza Help di MATLAB di `dblquad`, `integral2`, `integral3`, `quad2d`, `quadgk`, `quadv`, `trapz`, `triplequad`.

## 1.2 Cell-array e strutture con campi

I **cell-array** sono array che contengono dati di *tipo* e *dimensione differente*. Ad esempio, per memorizzare tre vettori di lunghezza diversa $\mathbf{v}_1 \in \mathbb{R}^4, \mathbf{v}_2 \in \mathbb{R}^2$ e $\mathbf{v}_3 \in \mathbb{R}^3$ nella stessa variabile $\mathbf{v}$, si *inizializza* una variabile di tipo cell-array con il comando `cell(n,m)`.

```
%inizializziamo un cell-array
v = cell(3,1)
%quindi memorizziamo i tre vettori
v{1}=[8; 0; -2; -1];
v{2}=[5; -6];
v{3}=[3; 0; 1];
```

Per leggere il contenuto della cell-array `v`, basta chiamare `v` stessa, mentre per visulizzare una componente particolare, ad esempio la seconda, basta chiamare `v{2}`.

Così come un cell-array, anche una **struttura** con **campi** e **valori** può contenere variabili di tipo diverso. Ad esempio, se vogliamo memorizzare una matrice quadrata, il suo determinante ed il numero di elementi non nulli in una sola variabile $A$, basta eseguire le istruzioni

```
% memorizziamo il campo mat che coniene la matrice
A.mat=[1 0 3; 2 -1 0; 0 1 2];
% memorizziamo il campo det che contiene il determinante
A.d=det(A.mat);
% memorizziamo il campo nz che contiene gli elementi
A.nz=nnz(A.mat);
```

Alternativamente, una struttura può essere definita in MATLAB con il comando `struct`. Per lo stesso esempio, possiamo richiamare le istruzioni

```
A=struct('mat',[1 0 3; 2 -1 0; 0 1 2],'d',4,'nz',6);
```

## 1.3 Formule di quadratura Gaussiane

Di particolare interesse nell'integrazione numerica sono le formule di quadratura che raggiungono un *grado di esattezza massimo*. Queste sono le *solo e uniche* formule di quadratura

- ⋆ che sono di tipo interpolatorie come definite tramite la (4)
- ⋆ i cui nodi sono i **nodi di Gauss**, ovvero gli **zeri** dei **polinomi ortogonali** definiti dalla (10)

e vengono chiamate **formule di quadratura Gaussiane**. Introduciamo brevemente i polinomi ortogonali. Supponiamo che l'intervallo in considerazione sia $[-1, 1]$. Considereremo integrali pesati della forma $\int_{-1}^{1} f(x)w(x)\,dx$, dove $w(x) > 0$ è una funzione peso, ovvero una funzione *non negativa* e *integrabile* su $[a, b]$. Si può introdurre un **prodotto scalare** ed una **norma** associati al peso $w$ tramite le definizioni

$$(f, g)_w = \int_{-1}^{1} f(x)g(x)w(x)\,dx, \quad \|f\|_w^2 = (f, f)_w. \tag{9}$$

Siamo interessati a considerare una successione $\{p_0, p_1, \dots\}$ di polinomi $p_k \in \mathbb{P}_k$, ortogonali rispetto a $(\bullet, \bullet)_w$, ovvero tali che $(p_i, p_j)_w = 0$ se $i \neq j$. Il seguente esempio fornisce un modo per costruirli.

**Esempio 1.1** (Polinomi ortogonali)**.** *Sia* $p_{-1}(x) = 0$, $p_0(x) = 1, \beta_0 \in \mathbb{R}$. *Si ponga*

$$p_k(x) = (x - \alpha_k)p_{k-1}(x) - \beta_{k-1}p_{k-2}(x), \ \textit{per } k = 1, 2, \dots \tag{10}$$

*dove*

$$\alpha_k = \frac{(xp_{k-1}(x), p_{k-1}(x))_w}{(p_{k-1}(x), p_{k-1}(x))_w}, \ \ \beta_k = \frac{(p_k(x), p_k(x))_w}{(p_{k-1}(x), p_{k-1}(x))_w}. \tag{11}$$

*Allora si può dimostrare che* $\{p_0, p_1, \dots\}$ *è una successione di polinomi monici ortogonali rispetto a* $(\bullet, \bullet)_w$.

Gli zeri dei polinomi ortogonali (10) vengono chiamati **nodi di Gauss**. Se il peso considerato nella (9) è

$$w(x) = \frac{1}{\sqrt{1 - x^2}} \tag{12}$$

troviamo i **polinomi di Chebyshev**, mentre per il peso

$$w(x) = 1 \tag{13}$$

troviamo i **polinomi di Legendre**. I loro **zeri**, sono detti rispettivamente **nodi di Gauss-Chebyshev** e **nodi di Gauss-Legendre**. Formule di integrazione interpolatorie del tipo (4) costruite rispetto a questi nodi sono dette **formule Gaussiane**.

Vale il seguente risultato fondamentale: il massimo grado di esattezza di una formula di quadratura $I_n$ a $n + 1$ nodi è $\boldsymbol{2n + 1}$. Questo si raggiunge quando

1. i nodi $x_i$, $i = 0, \dots, n$ sono i nodi di Gauss, ovvero gli zeri dell'$n + 1$-esimo polinomio ortogonale $p_{n+1}$ della successione (10);

2. i pesi sono dati $\int_{-1}^{1} \ell_i(x)\,dx$, essendo $\ell_i$ l'i-esimo polinomio della base di Lagrange associata ai nodi di Gauss.

Questo risultato vale per le formule **aperte**, ossia per cui i nodi di integrazione non comprendono gli estremi $-1$ e $1$. È possibile costruire formule **chiuse** tali che $x_0 = -1$ e $x_1 = 1$, che vengono chiamate di **Gauss-Lobatto**. Chiaramente, siccome il fatto di fissare due nodi fa perdere loro due gradi di esattezza, il loro grado di esattezza è $2n - 1$.

**Esercizio 1.1** (Polinomi ortogonali di Gauss-Legendre)**.** *Costruire un codice per generare e visualizzare i polinomi $p_k(x)$ della successione ortogonale di polinomi di Legendre* (10) *(con peso $w(x) = 1$) e i loro zeri per $k = 1, 2, \ldots, n+1$, con $n = 10$ e $[a, b] = [-1, 1]$.*

**Soluzione:**

```
n = 10;
a = -1; b = 1;

% set di punti per i plot
z = linspace(a,b,100);
% rappresentazione del polinomio x=1*x^1 + 0*x^0 -> [1,0]
x = [1,0];
% rappresentazione del polinomio p_{-1}(x)=0
pmeno2=0;

% inizializziamo un cell-array p che conterra' la successione dei
% polinomi di Legendre p_0,p_1, p_2, ....

% il primo elemento della successione dei polinomi di Legendre e'
% p_0(x)=1, che lo rappresentiamo  tramite lo scalare 1
p{1}=1;

% costruiamo gli altri elementi della successione p_1, p_2, ....
% tramite un ciclo for

% notare che per k=1,2,3, ... p{k+1} ha grado k e rappresenta il
% polinomio p_{k}(x) della successione (6)
for k=2:n+2

  % definiamo alpha_k=(x*p_{k-1},p_{k-1})/(p_{k-1},p_{k-1})

  espr_num=conv(conv(x,p{k-1}), p{k-1});
  num=polyval(polyint(espr_num),b)-polyval(polyint(espr_num),a);
  espr_den=conv(p{k-1}, p{k-1});
  den=polyval(polyint(espr_den),b)-polyval(polyint(espr_den),a);
  alpha=num/den;

  % definiamo p_k=(x-alpha_k)*p_{k-1} - beta_{k-1}*p_{k-2}

  p{k}=conv(x,p{k-1})-alpha*[0,p{k-1}];

  % siccome p{k-1} ha un grado in meno rispetto a p{k}, la
  % notazione [0,p{k-1}] permette di svolgere i calcoli
  % correttamente mantenendo la corrispondenza tra i coefficienti
```

```
  % dello stesso grado

  if k>2
      p{k}=p{k}-beta*[0,0,pmeno2];
  end

   % aggiorniamo il polinomio successivo
   pmeno2 = p{k-1};

   % definiamo beta_k=(p_k,p_k)/(p_{k-1},p_{k-1})
   espr_num_beta=conv(p{k}, p{k});
   num_beta=polyval(polyint(espr_num_beta),b)
              -polyval(polyint(espr_num_beta),a);
   beta=num_beta/den;

   % Radici del polinomio
   nodes = roots(p{k});
   plot(z,polyval(p{k},z),nodes,zeros(size(nodes)),'o');
   %plot(z,polyval(p{k},z));
   %title(sprintf('Polinomio ortogonale p%i ',k-1));
   grid on;
   hold on;
   title('Polinomi ortog. di Legendre p_k(x), k=1,...,n+1')
end
```

Dai grafici ottenuti rappresentati nella Figura 1 si osserva che gli zeri dei polinomi costruiti sono tutti *distinti*, *reali* e contenuti in $[a, b] = [-1, 1]$.

In seguito riassumiamo alcuni principali comandi relativi ai polinomi.

| **comando** | **significato** |
|---|---|
| `y=polyval(p,x)` | `y` = valori di $p(x)$ |
| `z=roots(p)` | `z` = radici di $p$ tali che $p(z) = 0$ |
| `p=conv(p1,p2)` | `p` = coefficienti del polinomio $p_1 p_2$ |
| `[q,r]=deconv(p1,p2)` | `q` = coefficienti di $q$, `r` = coefficienti di $r$ tali che $p_1 = qp_2 + r$ |
| `y=polyder(p)` | `y` = coefficienti di $p'(x)$ |
| `y=polyint(p)` | `y` = coefficienti di $\int_0^x p(t)\,dt$ |

**Esercizio 1.2** (Grado di esattezza delle formule di quadratura di Gauss)**.** *In base al risultato teorico enunciato, i nodi di integrazione che danno grado di esattezza massimo sono gli zeri dell'ultimo polinomio ortogonale calcolato $p_{n+1}$. Essendo un polinomio monico, fattorizzato rispetto alle sue radici esso risulta essere*

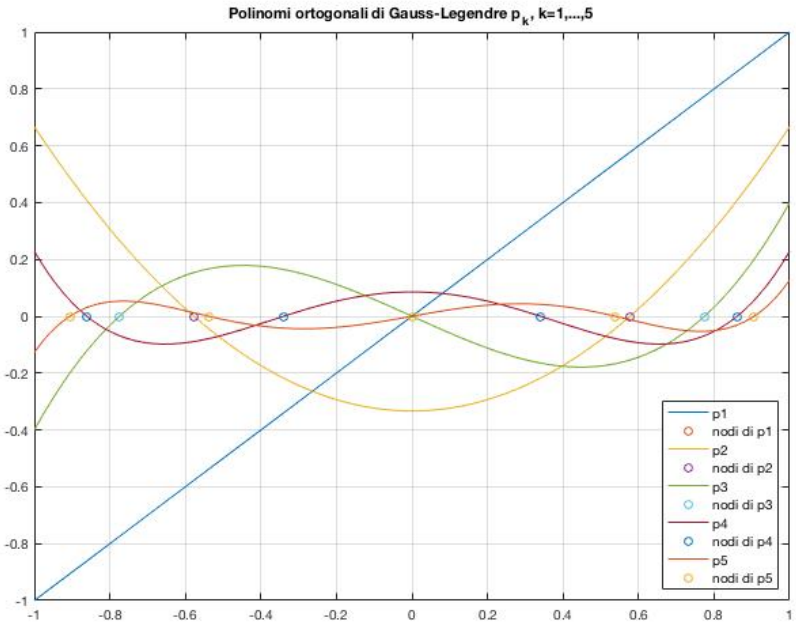



Figure 1: Polinomi ortogonali di Legendre $p_k(x)$, per $k = 1, \ldots, 5$

$$p_{n+1}(x) = (x - x_0)(x - x_1) \ldots (x - x_n) = \omega_{n+1}(x). \tag{14}$$

*Calcolare i pesi $w_i$ della formula di quadratura*

$$I_n(f) = \sum_{i=0}^{n} f(x_i) w_i, \tag{15}$$

*con*

$$w_i = \int_a^b \ell_i(x)\, dx, \tag{16}$$

*dove $\ell_i$ è il polinomio della base di Lagrange associato al nodo $x_i$. Sfruttare la formula compatta*

$$\ell_i(x) = \frac{\omega_{n+1}(x)}{(x - x_i)\omega'_{n+1}(x_i)} \tag{17}$$

*e i comandi* `deconv` *e* `polyder` *rispettivamente per la divisione fra polinomi e la derivazione di un polinomio. Dopodiché, calcolare l'errore di quadratura*

$$E_i = |I(x^i) - I_n(x^i)|, \quad \textit{per } i = 0, 1, \ldots, 2n + 2. \tag{18}$$

**Soluzione:**

```
% Polinomio nodale corrispondente all'ultimo polinomio ortogonale
% generato omega = p_{n+2} di grado n+1:
omega = p{n+2};

% Calcoliamo i pesi w(i), i=1,...,n+1 della formula di quadratura
% tramite un ciclo for che prima calcola i polinomi di Lagrange
% associati ai nodi x_i, li visualizza, poi li integra
% per ottenere i pesi

 for i=1:n+1
     % i-esimo polinomio della base di Lagrange
     % seguendo la formula suggerita
     [q, r] = deconv(omega , [1, -nodes(i)]);
     li = q/polyval( polyder(omega), nodes(i));

     plot(z, polyval(li, z), nodes , zeros(size(nodes)), 'o');
     title(sprintf('Polinomio della base di Lagrange l%i', i));
     pause;

     % i-esimo peso della formula di quadratura
     w(i) = polyval(polyint(li), b) - polyval(polyint(li), a);
 end

 % Errore di quadratura, testiamo la formula sui monomi x^i:
 for i=0 : (2*n + 2)
     % defininiamo il monomio x^i
     p = zeros(1, i+1); p(1) = 1;

     % valutiamo l'integrale esatto del monomio
     I = polyval(polyint(p), b) - polyval(polyint(p), a);
     % applichiamo la formula di quadratura al monomio
     In = w * polyval(p, nodes );
     % calcoliamo l'errore di quadratura
     e(i+1) = abs(I - In);
  end
```

Osservando i risultati del vettore `e`, notiamo come l'ultima componente dell'errore, corrispondente a $x^{2n+2}$, mentre stiamo usando $n = 10$, seppure piccola è comunque 7 ordini di grandezza maggiore di tutte le altre. La formula dunque è esatta di grado $2n + 1$, come ci si aspettava dalla teoria.

# 2 Approssimazione numerica di equazioni differenziali ordinarie

Un'**equazione differenziale** è un'equazione che coinvolge una o più derivate di una funzione incognita. Se tutte le derivate sono fatte rispetto ad una sola variabile indipendente avremo un'equazione differenziale **ordinaria** (ODE), mentre avremo un'equazione alle derivate parziali quando sono presenti derivate rispetto a più variabili indipendenti. L'equazione differenziale (ordinaria o alle derivate parziali) ha ordine $p$ se $p$ è l'ordine massimo delle derivate che vi compaiono.

Poniamo l'attenzione sui **problemi di Cauchy** che ricercano $y : I \subset \mathbb{R} \to \mathbb{R}$ tale che

$$\begin{cases} y'(t) = f(t, y(t)) \quad \forall t \in I, \\ y(t_0) = y_0, \end{cases} \tag{19}$$

dove I è un intervallo, $f : I \times \mathbb{R} \to \mathbb{R}$ è una funzione assegnata e $y'$ indica la derivata di $y$ rispetto a $t$. Infine, $y_0$ è un valore assegnato al tempo $t_0 \in I$ detto **dato iniziale**.

Siamo interessati a costruire **metodi numerici** in grado di approssimare la soluzione di ogni classe di ODE che ammettano una soluzione. La strategia generale di questi metodi consiste nel

- ⋆ dividere l'intervallo di integrazione $I = [t_0, T]$, con $T < \infty$, in $N_h$ sottointervalli di ampiezza $h = (T - t_0)/N_h$, dove h è detto il **passo di discretizzazione**;
- ⋆ per ogni nodo $t_n = t_0 + nh$ per $n = 1, \ldots, N_h$, cercare il valore incognito $u_n$ che approssimi $y_n = y(t_n)$.

La **soluzione numerica** è data dall'insieme dei valori $\{u_0 = y_0, u_1, ..., u_{N_h}\}$.

## 2.1 Metodo di Eulero

Un metodo classico è dato dal **metodo di Eulero**, che ha due versioni: il metodo di Eulero **in avanti** e il metodo di Eulero **all'indietro**.

Il metodo di Eulero in avanti genera la successione

$$u_{n+1} = u_n + hf(t_n, u_n), \quad n = 0, \ldots, N_{h-1}. \tag{20}$$

Questo metodo è derivato dall'equazione differenziale (21) considerata in ogni nodo $t_n$ con $n = 1, \ldots, N_h$, qualora si approssimi la derivata esatta $y'(t_n)$ con il **rapporto incrementale**.

```
function [t,u]=feuler(odefun ,tspan ,y0,Nh)
% Questa funzione risolve equazioni differenziali
% usando il metodo di Eulero in avanti.
% tspan = [t0,tf]
% la funzione integra il sistema di equazioni
```

```
% differenziali y'= f(t,y) dal tempo t0 a tf con
% condizione iniziale yo usando il metodo di Eulero
% in avanti su una griglia equispaziata di NH
% intervalli.
% La funzione odefun(t,y) deve ritornare un vettore
% contenente f(t,y), della stessa dimensione di y.
% Ogni riga del vettore soluzione y corrisponde ad
% un istante temporale del vettore colonna t.
h=(tspan(2)-tspan(1))/Nh;
t=linspace(tspan(1),tspan(2),Nh+1)';
y0=y0(:); % genera sempre un vettore colonna
d=length(y0);
u=zeros(Nh+1,d);
u(1,:)=y0.'; % trasposta anche di variabili complesse
for n=1:Nh
    w_n=u(n,:).';
    w=wn+h*odefun(t(n),wn);
    u(n+1,:)=w.';
end
```

**Esercizio 2.1.** *Considerare il problema di Cauchy*

$$\begin{cases} y'(t) = cos(2(y(t)) \quad \forall t \in (0,1] \\ y(0) = 0, \end{cases} \tag{21}$$

*la cui soluzione è* $y(t) = \frac{1}{2}arcsin((e^{4t}-1)/(e^{4t}+1))$. *Risolvere con il metodo di Eulero in avanti appena implementato usando diversi valori di* $h, h = 1/2, 1/4, 1/8, ..., 1/512$. *Calcolare il massimo errore max* $|y_n - u_n|$ *per ogni* $h$ *ed infine calcolare l'ordine di convergenza del metodo.*

```
tspan=[0,1];
y0=0;
f=@(t,y) cos(2*y);
u=@(t) 0.5*asin((exp(4*t) -1)./(exp(4*t)+1));
Nh=2;
for k=1:10
      [tn,un]=feuler(f,tspan ,y0,Nh);
      fe(k)=max(abs(un-u(tn)));
      Nh = 2*Nh;
end
```

Ultima versione aggiornata: 10 Giugno 2020

Link alle lezioni precedenti:
Lezioni 16-18
Lezioni 13-15
Lezioni 11-12
Lezioni 9-10
Lezioni 7-8
Lezioni 5-6
Lezioni 3-4
Lezioni 1-2

# Referenze

1. MATLAB® The language of technical computing: computation, visualization, programming, The MathWorks Inc
2. MATLAB: An introduction with applications, A. Gilat, Wiley
3. Scientific Computing with MATLAB and Octave, A. Quarteroni, , F. Saleri, P. Gervasio, Springer
4. MATLAB Guide1 D. J. Higham and N. J. Higham, SIAM
5. Numerical Computing with MATLAB, C. Moler, SIAM