Interpolazione e integrazione numerica

Carlo Mancini

16 dicembre 2015

Interpolazione con i polinomi di Lagrange

Integrazione numerica

Interpolazione con i polinomi di Lagrange - Introduzione

- Immaginiamo di conoscere il valore di $f(x)$ solo in alcuni punti $\{x_1, \ldots, x_n\}$
- Con l'interpolazione cerchiamo una funzione $L(x, \vec{c})$ (dove $\vec{c}$ sono una serie di parametri) che approssimi $f(x)$:

$$L(x, \vec{c}) \approx f(x) \tag{1}$$

  almeno nell'intervallo $[x_{min}, x_{max}]$
- Inoltre chiediamo che $L(x, \vec{c})$ eguagli $f(x)$ nei punti in cui quest'ultima assume un valore noto ($\{xi\}$)

$$L(x_i, \vec{c}) = f(x_i) \qquad \forall i \in \{1, \ldots, n\} \tag{2}$$

Interpolazione con i polinomi di Lagrange

- Possiamo cercare $L(x_i, \vec{c})$ come combinazione lineare di altre funzioni $\phi_i(x)$:

$$L(x_i, \vec{c}) = \sum_{i=1}^{n} c_i \phi_i(x) \tag{3}$$

  dove gli c_i sono dei coefficienti
- Grazie all'arbitrarietà su $L(x_i, \vec{c})$ e dunque sulle $\phi_i(x)$, possiamo imporre che:

$$\phi_i(x_j) = \delta_{ij} \tag{4}$$

- dove δ_{ij} è detta *delta di Kronecher* e vale 1 se $i = j$ e 0 altrimenti
- la condizione 4 significa chiedere che ogni ϕ_i valga 1 in $x = x_i$ e che in tutti gli altri punti x_j in cui è nota la funzione $f(x)$ valga 0; questa condizione ancora lascia arbitrarietà sulle ϕ_i

Interpolazione con i polinomi di Lagrange

- Ricordando che abbiamo imposto che $L(x, \vec{c})$ assuma lo stesso valore di $f(x)$ nei punti x_i in cui quest'ultima è nota

$$f(x_j) = L(x_j, \vec{c}) \tag{5}$$

- sostituendo a $L(x_i, \vec{c})$ la combinazione di funzioni più semplici $\phi_i(x)$:

$$f(x_j) = \sum_{i=1}^{n} c_i \phi_i(x_j) \tag{6}$$

- ricordando la condizione sulle ϕ_i nei punti in cui f è nota:

$$f(x_j) = \sum_{i=1}^{n} c_i \delta_{ij} \tag{7}$$

- possiamo fissare il valore dei coefficienti c_j

$$f(x_j) = c_j \tag{8}$$

Ricapitolando - Interpolazione con i polinomi di Lagrange

- $L(x_i)$ è una combinazione lineare di funzioni ϕ_i. I coefficienti della c.l. sono i valori della $f(x)$ in ciascuno dei punti x_i in cui assume un valore noto:

$$L(x) = \sum_{i=1}^{n} f(x_i)\phi_i(x) \tag{9}$$

- le ϕ_i sono ancora arbitrarie ma abbiamo imposto che ognuna valga 0 in tutti i punti x_j in cui $f(x)$ assume un valore noto eccetto che nel punto x_i, dove ϕ_i vale 1:

$$\phi_i(x_j) = \delta_{ij} \tag{10}$$

Interpolazione con i polinomi di Lagrange

- possiamo scegliere le ϕ_i come dei polinomi:

$$\phi_i(x) = \alpha \prod_{k=1, k\neq i}^{n} (x - x_k) \tag{11}$$

  dove gli x_k sono sempre i punti in cui $f(x)$ è nota
- questa scelta verifica la condizione $\phi_i(x_j) = \delta_{ij}$, infatti per ogni $x_j \neq x_i$ almeno uno dei termini della produttoria è nullo.

Interpolazione con i polinomi di Lagrange

- per imporre che ogni $\phi_i(x)$ valga 1 in $x = x_i$:

$$\phi_i(x_i) = \alpha \prod_{k=1, k\neq i}^{n} (x_i - x_k) = 1 \tag{12}$$

- da cui:

$$\alpha = \prod_{k=1, k\neq i}^{n} \frac{1}{(x_i - x_k)} \tag{13}$$

- quindi ogni ϕ_i:

$$\phi_i(x) = \prod_{k=1, k\neq i}^{n} \frac{(x - x_k)}{(x_i - x_k)} \tag{14}$$

Riassumendo - Interpolazione con i polinomi di Lagrange

- Ognuna delle ϕ_i è un polinomio di grado $n-1$

$$\phi_i(x) = \prod_{k=1, k\neq i}^{n} \frac{(x - x_k)}{(x_i - x_k)} \tag{15}$$

Interpolazione con i polinomi di Lagrange

- Infine:

$$L(x) = \sum_{i=1}^{n} f(x_i) \prod_{k=1, k\neq i}^{n} \frac{(x - x_k)}{(x_i - x_k)} \tag{16}$$

Esempi - Interpolazione con i polinomi di Lagrange

- Per $n = 2$ si parla di interpolazione lineare, infatti $L(x)$ è di primo grado:

$$L(x) = f(x_1)\frac{(x - x_2)}{(x_1 - x_2)} + f(x_2)\frac{(x - x_1)}{(x_2 - x_1)} \tag{17}$$

- Per $n = 3$:

$$\begin{aligned} L(x) &= f(x_1)\frac{(x - x_2)}{(x_1 - x_2)}\frac{(x - x_3)}{(x_1 - x_3)} && (18)\\ &+ f(x_2)\frac{(x - x_1)}{(x_2 - x_1)}\frac{(x - x_3)}{(x_2 - x_3)} && (19)\\ &+ f(x_3)\frac{(x - x_1)}{(x_3 - x_1)}\frac{(x - x_2)}{(x_3 - x_2)} && (20) \end{aligned}$$

Il codice - Interpolazione con i polinomi di Lagrange

La seguente funzione restituisce il valore di $L(y)$ dati gli array dei valori che la funzione assume (`f`) nei punti x

```
double Lagrange(double y, double *f, double *x, int n)
{
  double P, S = 0.;
  int i, j;

  for (i = 0; i < n; i++)
    {
      P = 1.;
      for (j = 0; j < n; j++)
        {
          if (j != i)
            {
              P *= (y - x[j]) / (x[i] - x[j]);
            }
        }
      S += P*f[i];
  }
  return S;
}
```

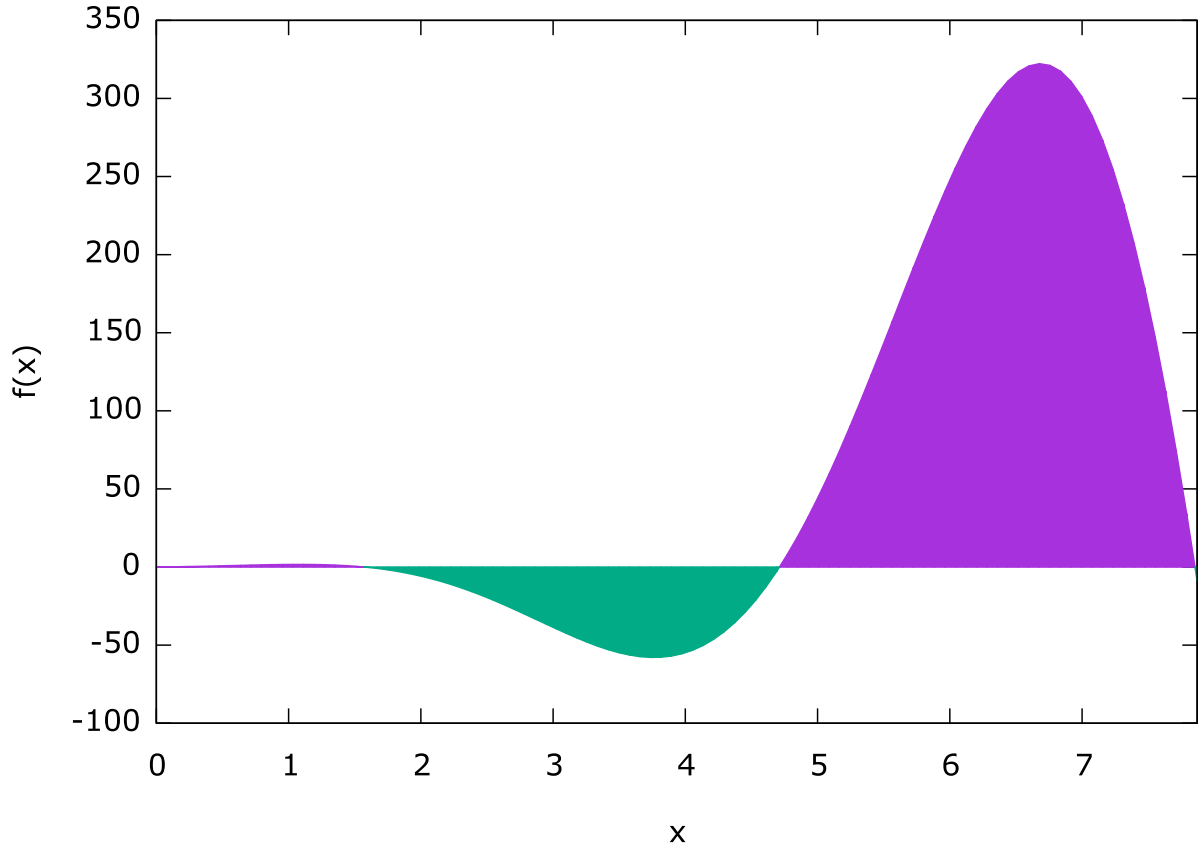
$$L(y) = \sum_{i=1}^{n} f(x_i) \prod_{j=1, j \neq i}^{n} \frac{(y - x_j)}{(x_i - x_j)}$$

```
double Lagrange(double y, double *f, double *x, int n)
{
  double P, S = 0.;
  int i, j;

  for (i = 0; i < n; i++)
    {
      P = 1.;
      for (j = 0; j < n; j++)
        {
          if (j != i)
            {
              P *= (y - x[j]) / (x[i] - x[j]);
            }
        }
      S += P*f[i];
  }
  return S;
}
```

Integrale di una funzione

- L'integrale è un operatore che data una funzione ne associa l'area sottesa dal grafico in un intervallo $[a, b]$

$$I \equiv \int_a^b f(x)dx \tag{21}$$



Primitiva di una funzione

- La primitiva di una funzione $f(x)$ è una $F(x)$ tale che:

$$F'(x) = f(x) \tag{22}$$

- allora l'integrale I di $f(x)$ è:

$$I \equiv \int_a^b f(x)dx = F(b) - F(a) \tag{23}$$

Additività degli integrali

- L'integrale di una funzione in un intervallo $[a, b]$ è uguale alla somma degli integrali negli intervalli $[a, c]$ e $[c, b]$ se $a < c < b$:

$$I \equiv \int_a^b f(x)dx = \int_a^c f(x)dx + \int_c^b f(x)dx \qquad (24)$$

Integrali numerici

- Non sempre la primitiva della funzione integranda è calcolabile analiticamente
- in questi casi si possono usare metodi numerici per stimare l'integrale

Sfruttare l'additività

- Possiamo dividere il dominio di integrazione $[a, b]$ in sottointervalli piccoli a piacere nei limiti imposti dall'architettura del computer $[c_0, c_1], [c_1, c_2], \ldots, [c_{n-1}, c_n]$
- a coincide con c_0 e b con c_n
- sfruttando l'additività degli integrali possiamo calcolare l'integrale I come la somma degli integrali sugli intervalli $[c_i, c_{i+1}]$

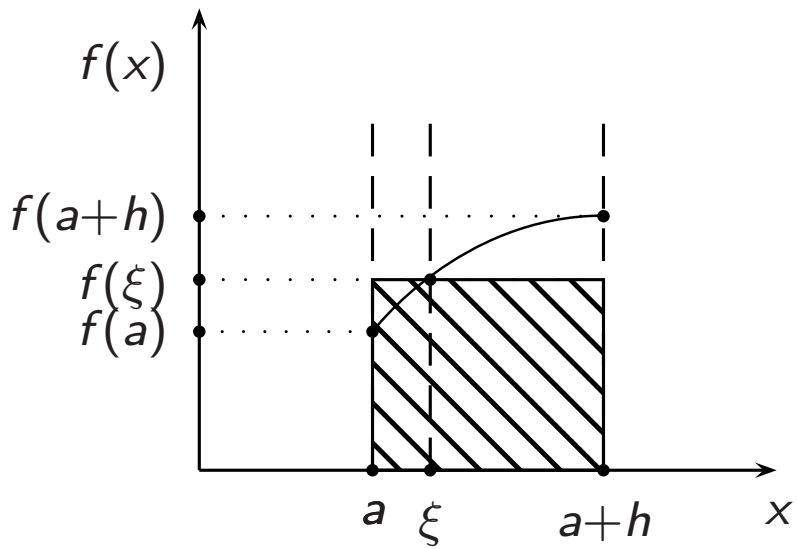
$$\int_a^b f(x)dx = \sum_{i=0}^{n-1} \int_{c_i}^{c_{i+1}} f(x)dx \tag{25}$$

Il metodo dei rettangoli

- se gli intervalli sono sufficientemente piccoli (e la funzione regolare) possiamo approssimare la funzione in ognuno degli intervalli come costante

$$\int_{c_i}^{c_{i+1}} f(x)dx = C_i \cdot (c_{i+1} - c_i) = C_i \cdot h \qquad (26)$$

dove h è la dimensione degli intervalli (ovvero: $c_{i+1} - c_i \ \ \forall i \in [0, n]$) e $C_i \equiv f(\xi)$, con $\xi \in [a, b]$ arbitrario

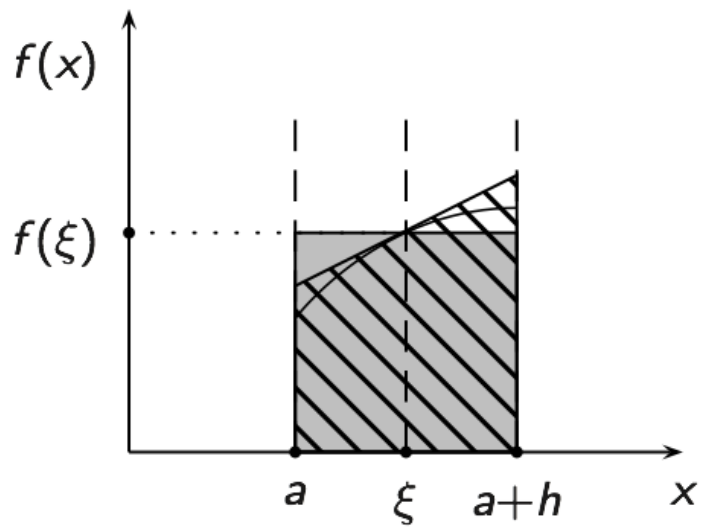


Il metodo del punto di mezzo

- si può scegliere come ξ il punto al centro di ogni intervallo

$$\xi = \frac{c_{i+1} + c_i}{2} \tag{27}$$

- equivalente a qualunque trapezio di altezza h, con il lato obliquo passante per $f(\xi)$



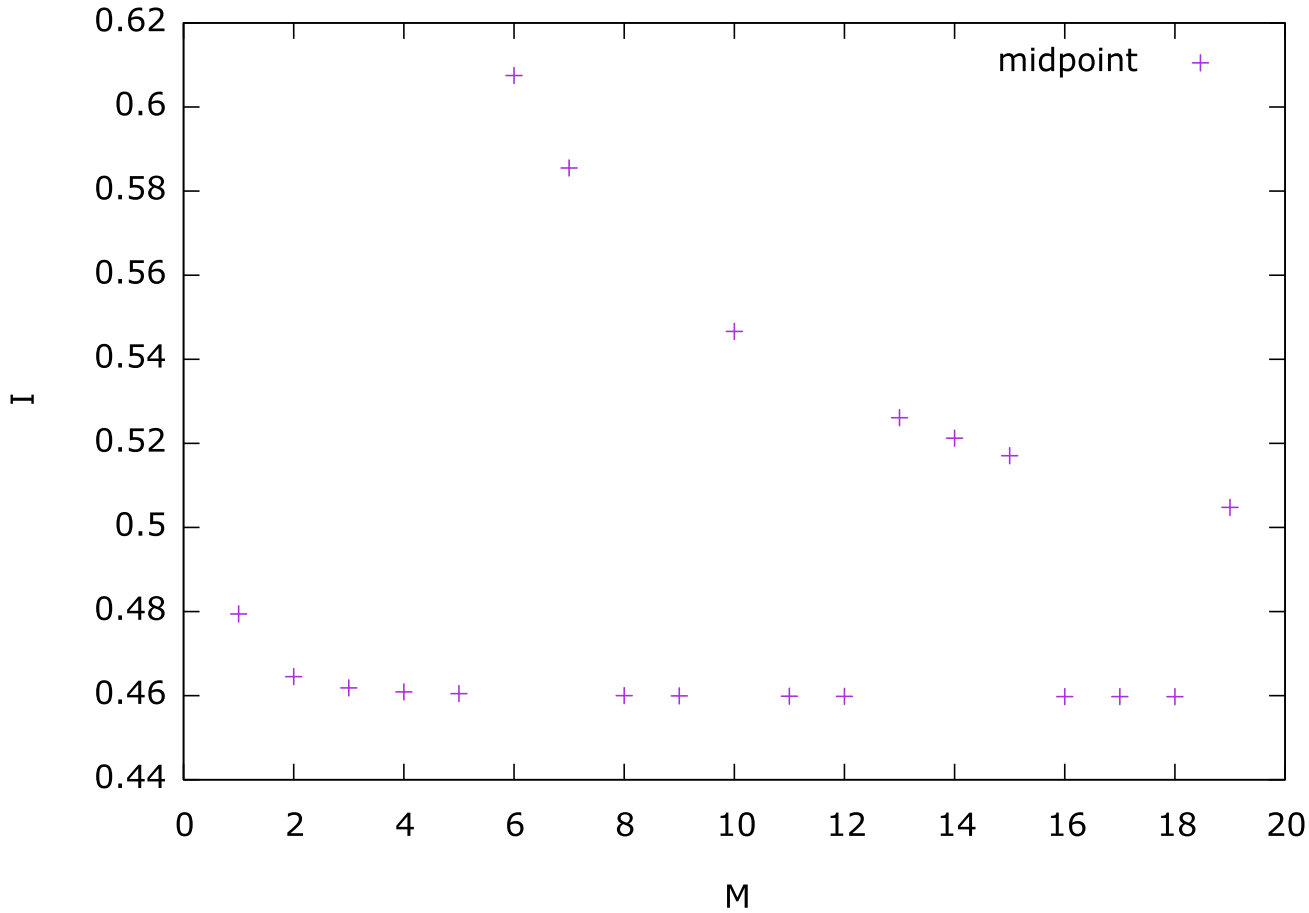
Il main

```c
#include <math.h>
#include <stdio.h>

#define TYPE double

TYPE midpoint(TYPE a, TYPE b, int n, TYPE (*f)(TYPE));
TYPE midpoint2(TYPE a, TYPE b, int n, TYPE (*f)(TYPE));
TYPE func(TYPE);

int main()
{
  TYPE a=0., b=2.M_PI;
  int M= 12;
  int i;
  TYPE I=midpoint(a,b,M,func);
  printf("I: \%lf \n",I);
  TYPE I2=midpoint2(a,b,M,func);
  printf("I2: \%lf \n",I2);
  return 0;
}
```

Una cattiva realizzazione dell'algoritmo del punto di mezzo

```
TYPE midpoint(TYPE a, TYPE b, int n, TYPE (*f)(TYPE))
{
  TYPE I = 0., x = a;
  TYPE h = (b - a) / (TYPE) n;

  int sc=0;

  while (x < b)
    {
      I += (*f)(x + 0.5 * h);
      x += h;
      sc++;
    }

#ifdef _DEBUG
  printf("midpoint h: \%.55f, n steps: \%d  \n", h, sc );
#endif

  return I * h;
}
```

Valore dell'integrale in funzione del numero di sottodomini

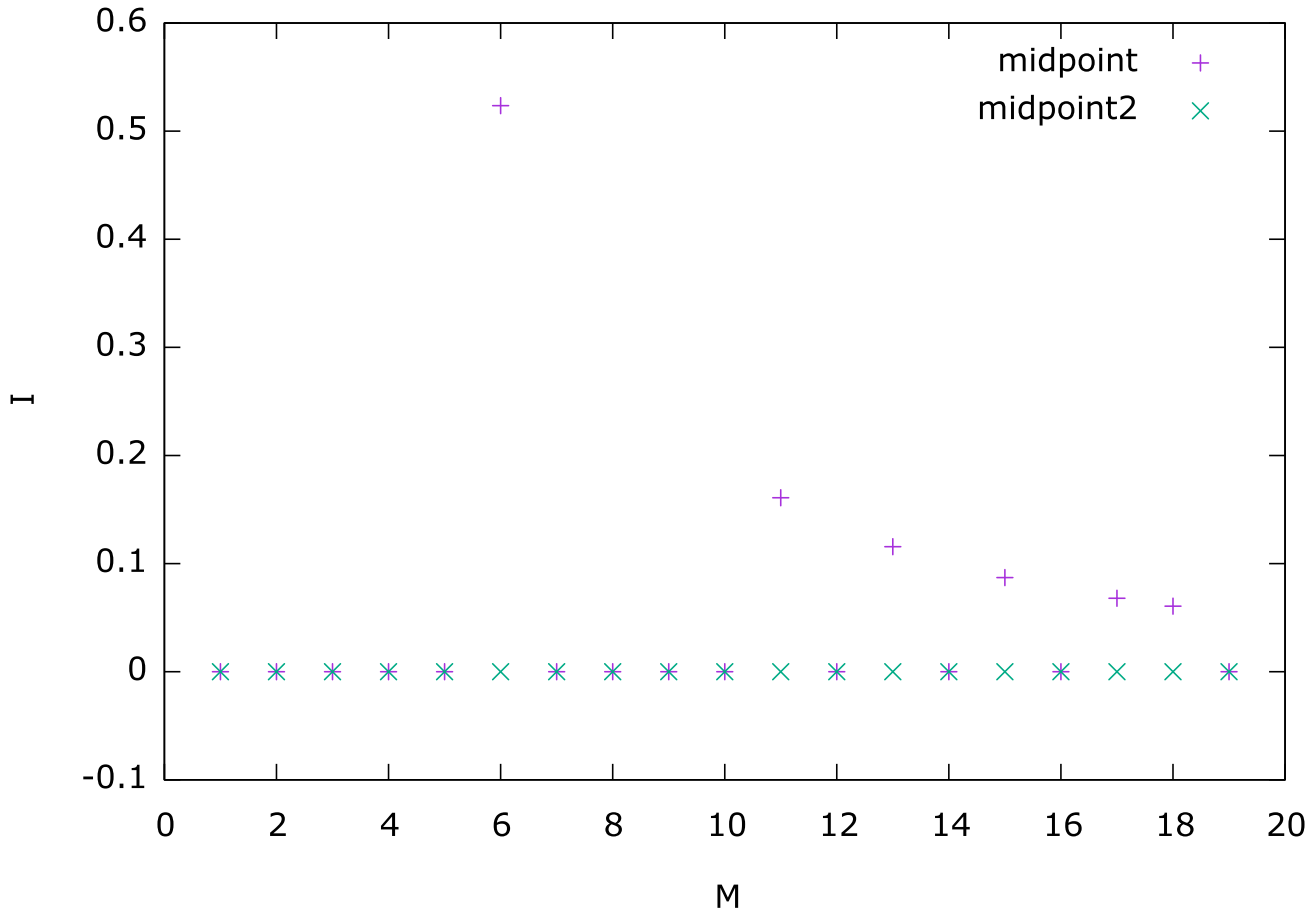


Integrale di $\sin(x)$ fra 0 e 1 in funzione del numero di sottodomini M con cui è calcolato. Sembrano esserci due diversi andamenti.

Una buona realizzazione dell'algoritmo del punto di mezzo

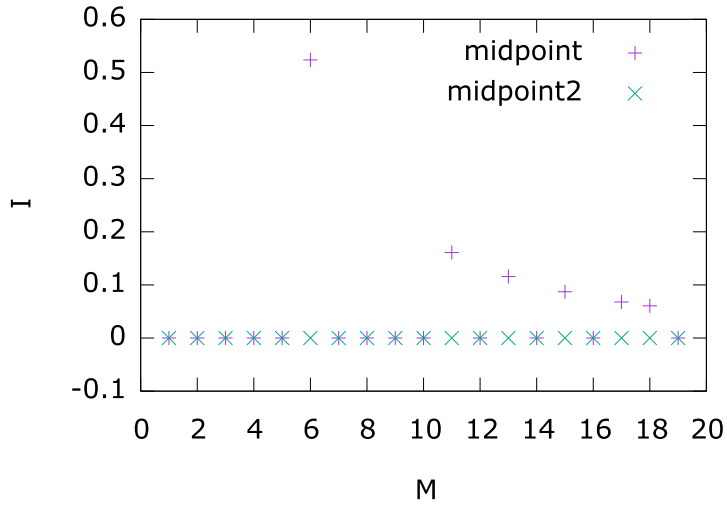
```
TYPE midpoint2(TYPE a, TYPE b, int n, TYPE (*f)(TYPE))
{
  TYPE I = 0., x;
  TYPE h = (b - a) / (TYPE) n;
  int i;

  int sc=0;

  for (i = 0; i < n; i++)
    {
      x = a + h * i + 0.5 * h;
      I += (*f)(x);
      sc++;
    }

#ifdef _DEBUG
  printf("midpoint2 h: \%.55f n steps: \%d \n", h, sc );
#endif

  return I * h;
}
```

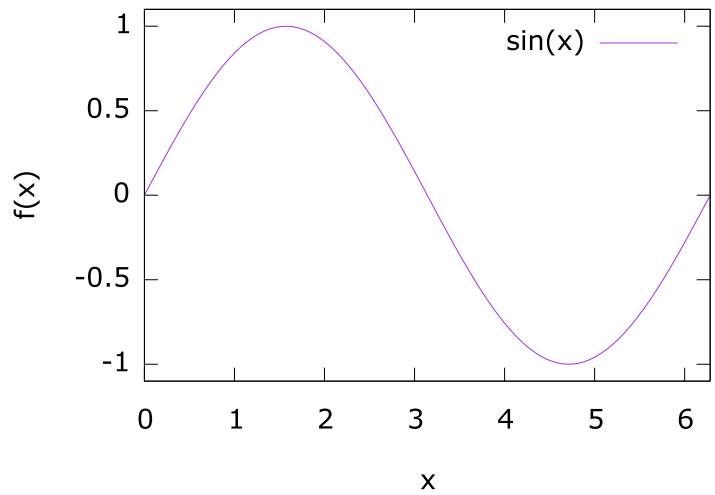
Valore dell'integrale in funzione del numero di sottodomini



Integrale di $\sin(x)$ fra 0 e 2π in funzione del numero di sottodomini M con cui è calcolato. Il primo algoritmo mostra la stessa anomalia, mentre il secondo no.

Valore dell'integrale in funzione del numero di sottodomini





(Anche ponendo $M = 1$ il valore dell'integrale viene 0 perché $\sin(x)$ è simmetrica rispetto al punto centrale del dominio di integrazione.)

Problemi di precisione

- 0.1 in rappresentazione binaria è:

- 0.1 è $1/10$. 10 in rappresentazione binaria è 1010. La divisione di 1 per 1010 fra binari è un numero periodico
- in rappresentazione decimale è un numero leggermente superiore o minore di 01. a seconda di dove viene troncata la rappresentazione

Problemi di precisione

- Un simile problema di precisione causa l'errore per alcuni valori di M nel primo algoritmo visto
- infatti il ciclo `while` termina quando `x` è maggiore o uguale a `b`
- all'ultimo passo `x` è uguale a `h` $\cdot$ `n` che, a meno di errori di precisione, è uguale a `b`
- tuttavia, proprio per i problemi di precisione esposti prima, per alcuni valori di M `x*h != b`, ed è più grande o più piccolo a seconda della macchina
- per questo motivo il primo algoritmo in questi casi fa un passo in più (o in meno) di quello che dovrebbe
- la seconda implementazione non è affetta da questo problema perché il ciclo `for` fa sempre `n` passi