TIPI PERSONALI

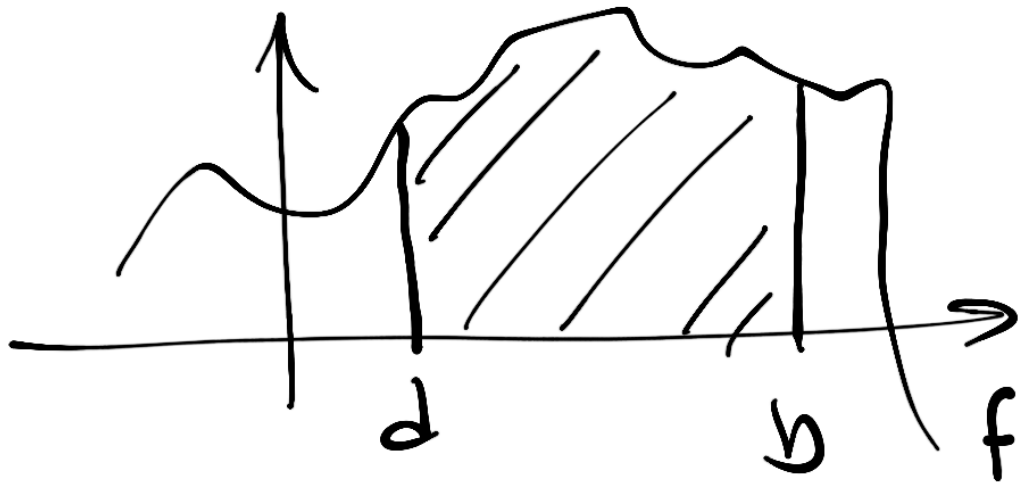
```
func Pippo(x int, y float64) bool

func(int, float64) bool
```

```
var f func(float64) float64
f = math.Sin
fmt.Println(f(1.0))
```

```
type funz func(float64) float64
var x []funz
x = []funz{math.Sin, math.Cos,
    math.Log}
for _, f := range x {
    fmt.Println(f(1.0))
}
```

$\int_a^b f(x)dx$

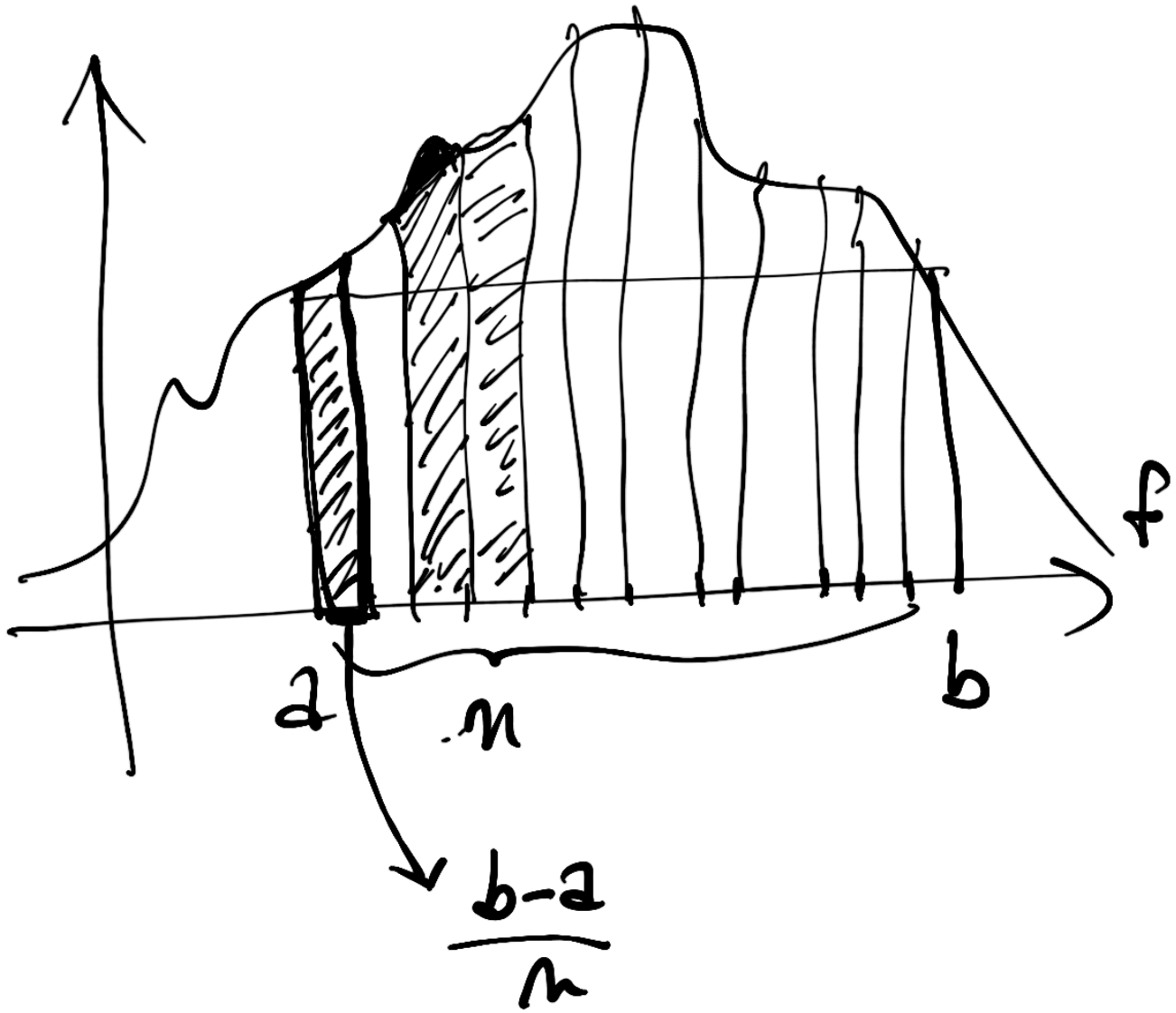


INT. SIMBOLICA

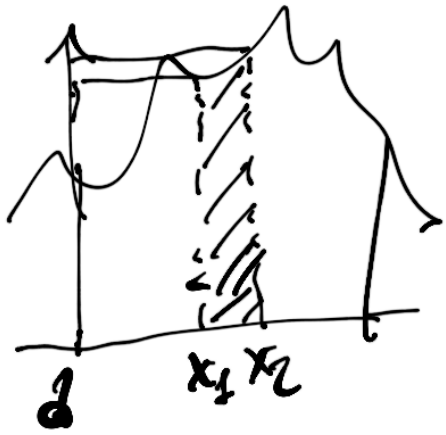
$F'(x) = f(x)$

$$\int_a^b f(x)dx = F(b) - F(a)$$

INT. NUMERICA


f
a
n
b
b-a
n

```
func integralTrap(f func(float64) float64,
a float64, b float64, n int) {
    var delta, s float64
    delta = (b-a)/float64(n)
    for i:=0; i<n; i++ {
        x1 := a + i*delta
        x2 := x1 + delta
        y1 := f(x1)
        y2 := f(x2)
        atrap := (y1+y2)*delta/2.0
        s += atrap
    }
    return s
}
```



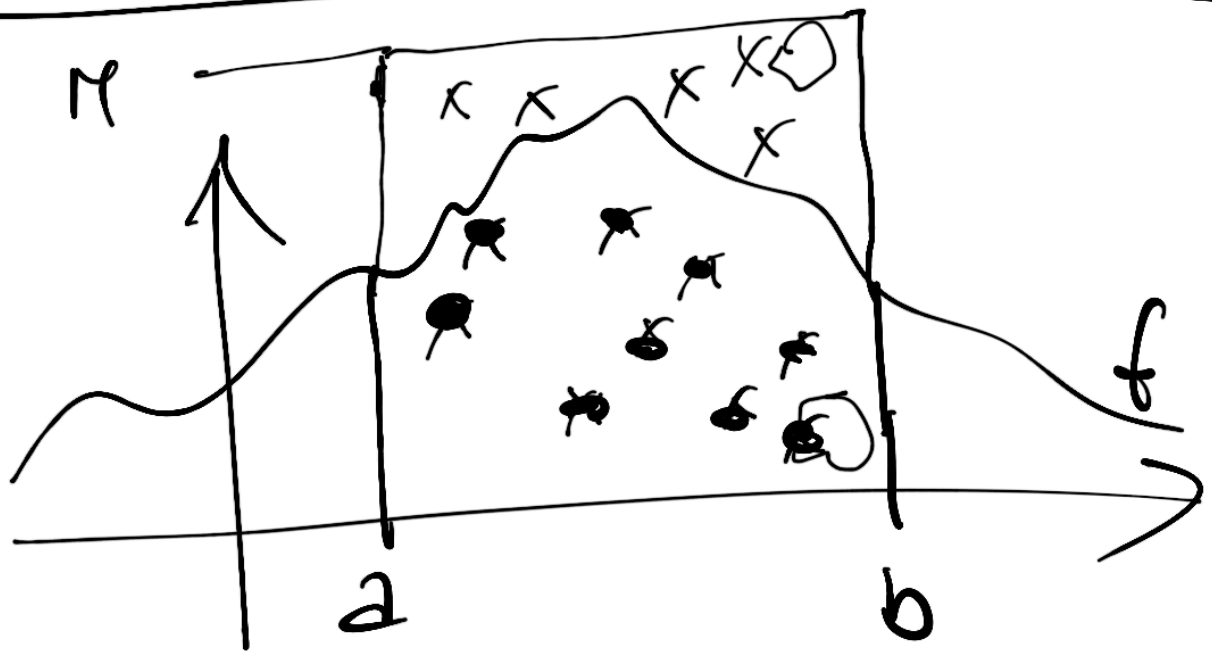
integraTrap(math.Sin, 0, 5, 1000)

$$\int_0^5 \sin(x)\,dx$$

$$\sin(x^2) + \cos^2(x)$$

```
integraTrap(func(x float64) float64 {
    return math.Sin(x*x) +
        math.Cos(x) * math.Cos(x)
}, 0, 5, 1000)
```

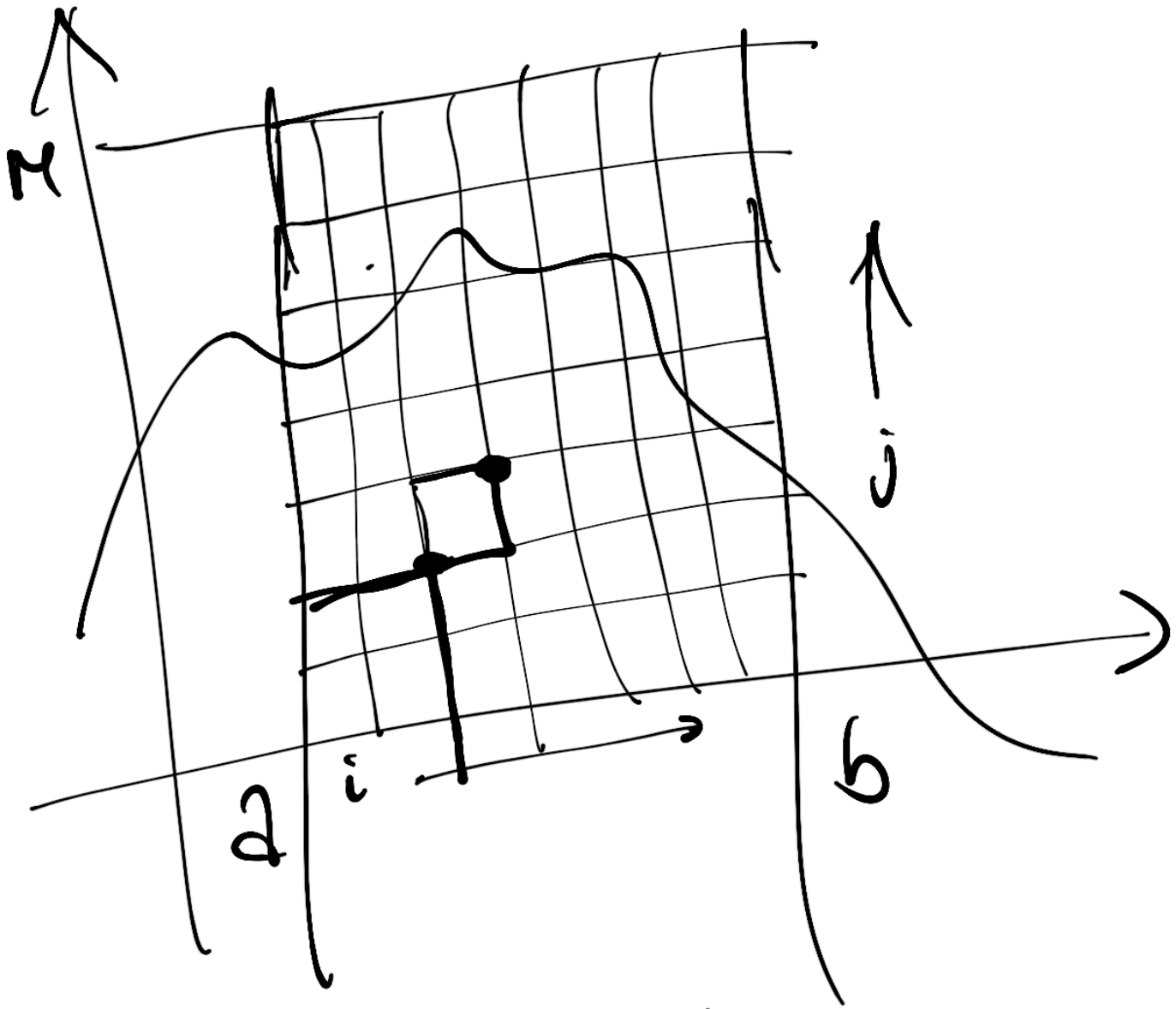
INTEGRALE MONTE-CARLO



```
func integra MC (func f (float64) float64,
a float64, b float64, M float64,
n int) float64 {
conta := 0
for i:=0; i<n; i++ {
    x := rand.Float64()*(b-a)
         +a
    y := rand.Float64()*M
    if y < f(x) {
        conta++
    }
}
return (b-a)*M*float64(conta)
       /float64(n)
}
```

```
func integraSimpson(f func(float64) float64,
a float64, b float64, M float64,
n int) float64 {

conta := 0
base := (b-a) / float64(n)
alt := M / float64(n)
for i := 0; i < n; i++ {
    for j := 0; j < n; j++ {
        bassoSx X := a + i * base
        bassoSx y := j * alt
        altoSx X := bassoSx X + base
        altoSx y := bassoSx y + alt
        x := bassoSx X + rand()
             * base
        y := bassoSx y + rand()
             * alt
        if y < f(x) {
            conta++
        }
```



a + i * base	+ base
j * alt	+ alt

```
import (
    "testing"
)

type funzione func(float64) float64

type testCase struct {
    f funzione
    F funzione
}

var testSuite []testCase {
    {math.Sin, func(x float64)
        float64 { return -math.Cos(x)
```

{sin x, -cos x} { ..., ... },
{ ..., ... }}

```
for _, tc := range testSuite {
    actual := integrateTrap(tc.f, 0, 5, 1000)
    expected := tc.F(5) - tc.F(0)
    if expected != actual {
        t.Error("Integrating ", tc,
            "expected", expected,
            "obtained", actual)
    }
}
```

math.Abs(expected - actual) < 1E-5