Numeri trascendenti

Vi siete mai chiesti quanti sono i numeri?

RESEARCH IN ACTION - RIA
RESEARCHINACTION.IT

Vi siete mai chiesti quanti sono i numeri? Infiniti: questa è la risposta!

I numeri possono essere: naturali, interi, razionali, reali o irrazionali.

Noi abbiamo lavorato sui numeri irrazionali e in particolare sui numeri trascendenti.

Ma, di preciso, cos'è un numero trascendente? I numeri trascendenti sono numeri irrazionali che non sono numeri algebrici e quindi, non sono soluzione di nessuna equazione polinomiale (con coefficienti interi o razionali).

11 Numeri trascendenti - 05.19
Revisione 0 del 27.05.19

math inside

RiA - Research in Action

La parola ria in inglese significa estuario, in particolare (dalla definizione che ne da l'Oxford Living Dictonaries):

> A long, narrow inlet formed by the partial submergence of a river valley ... the rias or estuaries contain very peculiar ecosystems which often contain important amounts of fish ... (a causa della loro natura, le rias o estuari contengono ecosistemi molto particolari che spesso contengono grandi quantità di pesce - www.eurotomic.com/spain/the-rias-altas-in-spain.php)

quindi questo prodotto che sarà realizzato grazie all'attività di alternanza scuola-lavoro di alcuni studenti del liceo scientifico G.B.Grassi di Latina - **www.liceograssilatina.org** - sarà un luogo virtuale da esplorare dove *pescare* molto materiale per la didattica laboratoriale.

Fare scienza

La scienza non è solo identificabile con la formula, il modello, la teoria. In altre parole la scienza non rappresenta solo un corpo di conoscenze organizzate e formalizzate. La scienza è anche e fondamentalmente ricerca. Una ricerca volta a conoscere e a capire sempre più e sempre meglio come è fatto e come funziona questo nostro complicatissimo mondo.

Fare scienza si identifica con l'interrogarsi, con l'indagare ed esplorare fatti e cose. Questo tipo di lavoro i bambini lo fanno spontaneamente sin dalla loro nascita ma si perde nel corso del percorso scolastico. L'intervento educativo deve tener conto di ciò e fornire stimoli, occasioni e strumenti per far acquisire agli studenti capacità sempre più ampie e affinate per poter compiere questo lavoro di indagine mantenendo viva (o risvegliando) la curiosità cognitiva, la voglia di sapere e di scoprire, la fiducia di poter capire.

Pensare in senso creativo, in campo scientifico, significa aggredire i problemi, attivare processi vivi dei pensiero, alimentare l'evoluzione dinamica dell'intelligenza duttile, dell'esercizio dell'intuizione e dell'immaginazione, della capacità di progettare e formulare ipotesi, di controllare e verificare quanto prodotto e ricercato.

Per questo è necessario bandire forme di apprendimento consumate entro schemi rigidi di elaborazione del pensiero e puntare al recupero della congettura, dell'ipotesi, di una coscienza scientifica aperta a interrogare ogni problematica.

La società odierna deve far fronte ad un rinnovamento scientifico e tecnico accelerato in cui lo sviluppo delle conoscenze scientifiche e la creazione di prodotti di alta tecnologia *(hi-tech)*, come anche la loro diffusione subiscono un'accelerazione sempre più rapida.

È necessaria, quindi, una diffusione della conoscenza in genere ed è indispensabile promuovere una nuova cultura scientifica e tecnica basata sull'informazione e sulla conoscenza. E quanto più è solida la base di conoscenze scientifiche scolastiche, tanto più si può approfittare dell'informazione e della conoscenza scientifica e tecnica.

» https://www.facebook.com/Research-in-Action-341307966417448/
» https://www.youtube.com/channel/UC1PA7Zu78RUMBJnkaiOR8kA/

Sommario dei contenuti

Numeri trascendenti - Vi siete mai chiesti quanti sono i numeri?

Sommario dei contenuti

1. Quanti sono i numeri? 5

1.1. Ma a cosa servono questi numeri trascendenti? 5
1.2. Cosa vi chiediamo di fare 5
1.3. Prerequisiti 6
1.4. Obiettivi 6

2. Si comincia! 7

2.1. Qualche termine che potrebbe tornare utile 7
2.2. La prima procedura: il primo successivo p 8
2.3. Fattoriale 9
2.4. Scomporre in fattori 10
2.5. Il metodo di Taylor per approssimare il numero di Eulero 13
2.6. Metodo di Newton per approssimare il numero di Eulero 15
2.7. Minimo scostamento 16

3. Soluzioni 18

3.1. La prima procedura: il primo successivo p 18
3.2. Fattoriale 19
3.3. Scomporre in fattori 20
3.4. Il metodo di Taylor per approssimare il numero di Eulero 21
3.5. Metodo di Newton per approssimare il numero di Eulero 22
3.6. Minimo scostamento 23

4. Ultimo step! 24

4.1. Radice quadrata di due 24
4.2. Pi greco 24
4.3. Il numero di Eulero 25

5. Esercizi 26

5.1. Codifica con Blockly 26
5.2. Estensione ad altri trascendenti 26
5.3. Pi greco, un'approssimazione migliore 27
5.4. Estensione ad altri irrazionali 27

6. La libreria 28

6.1. Approssimazione di numeri irrazionali e trascendenti 28
6.2. Derivata aritmetica 30

Blockly è un generatore di codice open source sostenuto da Google: https://developers.google.com/blockly/.

Materiale disponibile per questo laboratorio:

» il fascicolo (in formato PDF di circa 10MB): **http://researchinaction.it/wp-content/uploads/2019/05/11-Numeri-trascendenti.pdf**;

» la libreria, sviluppata con Blockly, si può provare sul campo: **http://researchinaction.it/myblockly/numbertheory.html** ed è descritta nel dettaglio qui: **http://researchinaction.it/2019/04/11/trascendenti-la-libreria/**.

Per il materiale didattico a supporto del fascicolo visitare anche la pagina Download del sito dedicato al progetto: **http://researchinaction.it/download/**.

Per i videotutorial è possibile visitare il canale YouTube del progetto: **https://www.youtube.com/channel/UC1PA7Zu78RUMBJnkaiOR8kA**.

Numeri trascendenti

Vi siete mai chiesti quanti sono i numeri?

1. Quanti sono i numeri?

Questo laboratorio è stato sviluppato da Francesca Arcese, Roberta Befera, Andrea Buzzacco, Sofia Corradetti, Ludovica Fiori, Alessandra Todesca in collaborazione con Donato Bini (CNR-IAC), il progetto è stato coordinato da Gualtiero Grassucci (Iss G.B. Grassi di Latina).

Vi siete mai chiesti quanti sono i numeri? Infiniti: questa è la risposta!

I numeri possono essere: naturali, interi, razionali, reali o irrazionali. Noi abbiamo lavorato sui numeri irrazionali e in particolare sui numeri trascendenti. Ma, di preciso, cos'è un numero trascendente? I numeri trascendenti sono numeri irrazionali che non sono numeri algebrici e quindi, non sono soluzione di nessuna equazione polinomiale (con coefficienti interi o razionali).

Per esempio: radice quadrata di due è un numero irrazionale ma non è trascendente, perché è soluzione dell'equazione

$$x^2 - 2 = 0$$

Infatti, isolando il termine di secondo grado in x si ha:

$$x^2 = 2$$

da cui, facilmente, le due soluzioni dell'equazione:

$$x = \pm\sqrt{2}$$

Nonostante sia numeri algebrici che trascendenti siano infiniti, questi ultimi sono infinitamente di più dei primi! Ebbene sì: esistono infiniti più grandi di altri infiniti!

Il famoso $\pi = 3.14159...$ (pi greco) per esempio, fa parte della famiglia dei numeri trascendenti! Infatti non è possibile scrivere un'equazione con coefficienti interi o razionali che ammetta, tra le proprie soluzioni, il numero pi greco. Sono numeri davvero speciali, con infinite cifre decimali che si ripetono in modo sempre diverso, combinandosi in tutti i modi possibili!

1.1. Ma a cosa servono questi numeri trascendenti?

Un esempio su tutti è emblematico. La scoperta dei numeri trascendenti consentì di dimostrare che certi problemi geometrici storici riguardanti la costruzione con riga, squadra e compasso erano impossibili. La quadratura del cerchio è forse il più famoso tra questi problemi. Esso consiste nel disegnare un quadrato della stessa area di un cerchio, ma visto che π (pi greco) è trascendente mentre tutti i numeri costruibili con riga e compasso sono algebrici, l'operazione è impossibile.

1.2. Cosa vi chiediamo di fare

Per prima cosa *fare amicizia* con questi numeri e con i loro sviluppi formali per arrivare ad approssimarli con rapporti polinomiali o altre funzioni.

Ma l'obiettivo veramente interessante e quindi anche più complesso (perché per noi bello è sinonimo di tosto!) è quello di realizzare una libreria di procedure che permetta a chiunque di giocare con i numeri trascendenti: calcolare gli sviluppi con tantissime cifre decimali, verificare somiglianze, immaginare teoremi …

1.3. Prerequisiti

Prima di iniziare il nostro laboratorio, è necessario:

» saper determinare una funzione che approssimi una serie di dati sperimentali;
» conoscere il concetto di derivata;

In alcuni casi è necessario utilizzare dei software CAS *(Computer Algebra System)*, come Geogebra. Le procedure che saranno realizzate nel corso del laboratorio possono essere implementate usando Blockly.

1.4. Obiettivi

Sul blog Research in Action è possibile utilizzare la libreria sviluppata con Blockly nel corso del laboratorio: http://researchinaction.it/myblockly/numbertheory.html
Blockly è un generatore di codice open source sostenuto da Google: https://developers.google.com/blockly/.

L'obiettivo è costruire, in linguaggio naturale ma abbastanza rigoroso, procedure per approssimare numeri irrazionali (come radice quadrata di due) o trascendenti (come π pi greco o il numero di Eulero *e*) e dare una stima della convergenza dei metodi trovati.

Obiettivo ulteriore è tradurre le procedure, scritte in linguaggio naturale, in un linguaggio di programmazione formale usando il generatore di codice Blockly. Blockly è un generatore di codice sviluppato da Google, *open source*, nato per facilitare l'apprendimento delle basi del *coding* e la robotica educativa. Unendo i blocchi di comando come i pezzi di un puzzle, si possono costruire funzioni via via più complesse.


```
3.14159265358979323846264338327950288419716939937510582097494459230781
64062862089986280348253421170679821480865132823066470938446095505822317
25359408128481117450284102701938521105559644622948954930381964428810975
66593344612847564823378678316527120190914564856692346034861045432664821
33936072602491412737245870066063155881748815209209628292540917153643678
92590360011330530548820466521384146951941511609433057270365759591953092
18611738193261179310511854807446237996274956735188575272489122793
```

2. Si comincia!

Chiarite le premesse, ora possiamo puntare al nostro obiettivo: progettare algoritmi e formalizzare procedure per approssimare i numeri trascendenti!

2.1. Qualche termine che potrebbe tornare utile

Durante la lettura del fascicolo potresti trovare *termini difficili*, come input, output o algoritmo. Con *input* e *output*, intendiamo rispettivamente il/i dato/i che immettiamo nella fase iniziale dell'algoritmo, e ciò che otteniamo alla fine, il risultato di un algoritmo.

Per esempio, nella procedura (nel calcolo) che permette di determinare le radici di un'equazione di secondo grado, per procedere, abbiamo bisogno di conoscere i coefficienti a, b e c dell'equazione della parabola: queste informazioni saranno l'input della nostra procedura. Le soluzioni dell'equazione saranno l'output, il risultato, le informazioni che volevamo conoscere (a partire da quelle fornite come input).

Ma cos'è veramente un algoritmo? È un procedimento sistematico di calcolo, che dopo aver fornito dei dati iniziali (input), restituisce uno o più risultati finali (output).

Proseguendo nell'analisi dell'esempio precedente, l'algoritmo sarebbe la formula risolutiva dell'equazione, magari con un controllo sul segno del discriminante.

Il Toolbox: http://researchinaction.it/wp-content/uploads/2018/11/00-Toolbox.pdf introduce i principali concetti e metodi per un approccio informale al coding e alla formalizzazione delle procedure e degli algoritmi (cfr. 6. Algoritmi a pagina 23).

Per controllare il corretto funzionamento di un algoritmo può essere utile lavorare con carta e penna: impostare una tabella con una colonna per ogni variabile della procedura e una riga per ogni istruzione eseguita e ripercorrere la procedura passo per passo, istruzione dopo istruzione, cercando di fare proprio quello che la procedura richiede. Per maggiori chiarimenti, è davvero utile consultare il Toolbox che amplia e precisa il discorso.

Tutto il laboratorio, come sarà chiaro in seguito, è basato sulla formalizzazione di procedure, la traduzione di un algoritmo, di un metodo, in un linguaggio semplice, non ambiguo, ma molto simile all'italiano, che chiameremo *pseudo-codice*.

Ah, quasi dimenticavamo! Leggendo questo fascicolo si incontrerà molto spesso la parola *iterazioni*, quindi mettiamo in chiaro che cosa sono. Le iterazioni, sono il numero di volte che si vuolesia fatta una certa cosa. Quindi, se ti viene chiesto di fornire alla funzione il numero di iterazioni, quello che dovrai inserire è un numero naturale intero.

Come procedere

In questo fascicolo guideremo il lettore usando un metodo *bottom-up*, partendo cioè dalle procedure e dagli algoritmi più semplici che saranno i *mattoncini* con cui costruiremo le funzioni che approssimeranno i numeri irrazionali e trascendenti: vero obiettivo del laboratorio. Questo percorso ha un innegabile vantaggio, se il lettore non ha familiarità con la formalizzazione delle procedure avrà la possibilità di prendere via via confidenza con l'argomento prima di affrontare i problemi *belli e tosti* che sono lo scopo finale. Ciò non toglie che è possibile tralasciare i dettagli (gli algoritmi basilari, funzionali alla soluzione del problema ma non essenziali dal punto di vista didattico) e affrontare direttamente le procedure più interessanti e gratificanti.

Nel testo, ogni tanto, sarà suggerita la possibilità di *saltare* un paragrafo, di cui si può fare a meno, per procedere in modo più rapido verso la meta.

2.2. La prima procedura: il primo successivo p

Per alcune delle funzioni necessarie all'approssimazione dei numeri trascendenti è necessario realizzare una funzione preliminare che sia in grado di trovare il numero primo successivo a un numero intero specificato. È una procedura piuttosto semplice ed è particolarmente utile per familiarizzare con i metodi che saranno usati in questo laboratorio ma, se hai già confidenza con gli algoritmi e/o vuoi concentrarti sul problema principale, puoi andare direttamente al paragrafo xx Il metodo di Taylor a pagina yy.

Con primo successivo intendiamo il numero primo p successivo ad un numero dato n. Per esempio, dato $n = 14$, il primo successivo è $p = 17$: il numero primo che si incontra subito dopo *14*.

Ma cos'è un numero primo? Un numero primo è un numero intero naturale e positivo che è divisibile solamente per 1 e per se stesso. Facile, no?

Ma praticamente come si fa a scrivere una procedura del genere? Dato un numero intero qualsiasi, bisogna aggiungere uno e domandarsi se il numero ottenuto è primo. Qualora lo sia, abbiamo trovato il primo successivo, altrimenti si continua ad aggiungere uno al numero ottenuto di volta in volta finchè non si arriva ad ottenere un numero primo, il primo successivo.

Un esempio

Il problema: trovare il primo successivo p di un numero intero n.

Ipotesi: Supponendo di dover trovare il primo successivo p del numero primo 7, la mia funzione dovrebbe restituire 11. Partendo dal primo numero (in questo caso 7) cerco il primo successivo: quindi procedo aggiungendo 1 a 7: 7+1=8, 8 è un numero primo? La risposta è no in quanto divisibile per 2. Quindi procedo aggiungendo 1 ad 8. *8+1=9*, 9 è un numero primo? La risposta è no. Continuo sommando ogni volta uno al numero appena trovato fino a quando non arrivo ad ottenere un numero primo p. Nel caso preso come esempio p sarà 11 (undici è un numero primo, infatti). Ho trovato il *primo successivo* a 7.

In termini un po' più formali, la cosa è andata più o meno così:

» Dati in input: $n = 7$
» $p = 7+1$ (p è uguale a 8)
» otto è primo? No, allora continua:
» $p = 8+1$ (p è ora uguale a 9)
» nove è primo? No, allora continua
» $p = 9+1$ (p adesso è pari a 10)
» 10 è primo? No, allora continua
» $p = 10+1$ (con p che è arrivato a 11)
» 11 è primo? Sì quindi arrestiamo il procedimento
» il nostro dato in output è 11

La procedura

Cerchiamo ora di formalizzare meglio il metodo che abbiamo usato. L'obiettivo è quello di scrivere alcune righe di testo in modo molto precisio, non ambiguo, perchè chiunque, anche all'oscuro dello scopo della procedura, possa raggiungere il risultato voluto senza bisogno di aiuto o altre spiegazioni.

Precisa l'input della procedura, in altre parole tutto quello che serve perchè la procedura esegua il calcolo. Per questo caso particolare, dovrebbe essere abbastanza facile.

La prima riga della nostra procedura (ma anche delle altre) costituirà l'input.

Una volta capito di cosa ha bisogno la procedura, possiamo passare all'algoritmo vero e proprio.

Analizza bene l'esempio precedente, ci sono alcuni passi che vengono ripetuti più volte. Riesci a scrivere in modo preciso (rigoroso) e generale (non dipendenti dal particolare numero scelto) questi passi e le operazioni da compiere?

Le frasi che hai scritto costituiscono il cuore della nostra (piccolissima) procedura. Queste frasi devono essere ripetute più volte fino a raggiungere l'obiettivo.

Specifica quali frasi, istruzioni, devono essere ripetute più volte e indica chiaramente una condizione da soddisfare perchè il ciclo (composto dalle istruzioni ripetute) si interrompa.

Se sei riuscito a costruire il ciclo come richiesto hai quasi completato il lavoro.

Specifica l'output dell'algoritmo, il risultato finale del processo.

Una procedura molto spesso ha bisogno di una fase di test, deve essere messa alla prova per essere sicuri della bontà delle scelte fatte.

Ora che hai completato la procedura, prova a usarla con altri valori.

Funziona? Se sì, abbiamo ottenuto la nostra prima procedura.

Se hai poca confidenza con questi meccanismi (definizione di un algoritmo, formalizzazione di una procedura, ...) può essere utile leggere le soluzioni di questo primo (piccolo) problema prima di proseguire. La nostra proposta per la funzione che determina il primo successivo a un intero dato è descritta nel paragrafo xx Il primo successivo p a pagina yy.

2.3. Fattoriale

Molti linguaggi di programmazione, generatori di codice o software CAS hanno una funzione predefinita per calcolare il fattoriale. Se questo è il tuo caso o se sei interessato agli obiettivi fondamentali del laboratorio e meno ai dettagli, puoi andare direttamente al paragrafo xx Metodo di Taylor che trovi a pagina yy.

Un'altra funzione assolutamente indispensabile al nostro lavoro è quella che calcola il fattoriale di un numero naturale n dato come input. In altre parole, un metodo per calcolare il prodotto di tutti i numeri naturali da 1 al numero n fissato, e restituire il prodotto ottenuto come output.

$$n! = 1 \cdot 2 \cdot \ldots \cdot n = \prod_{i=1}^{n} i$$

Per definizione il fattoriale di zero è uno, così come il fattoriale di uno. Quindi la nostra funzione, in caso l'input sia zero o uno deve restituire come output uno.

Per esempio, come si fa a calcolare il fattoriale di 4? Bisogna moltiplicare tutti i numeri naturali a partire da 1 fino ad arrivare n (nel nostro caso 4). In pratica, moltiplichiamo 1 per 2 ottenendo 2, poi moltiplichiamo il risultato (2) per 3 e otteniamo 6, infine moltiplichiamo ancora l'ultimo risultato per 4 e ricaviamo 24: il fattoriale di 4.

Ora proviamo a scrivere concretamente la procedura.

Precisa l'input della procedura, cioè tutto quello che serve perchè la procedura esegua

Blockly Google
A Visual Programming Language

Blockly non ha un blocco predefinito per il fattoriale, per cui se avete intenzione di implementare gli algoritmi sviluppati in questo laboratorio con Blockly è necessario aggiungere anche una funzione per il calcolo del fattoriale. A questo proposito vedere la libreria sviluppata proprio con Blockly: http://researchinaction.it/myblockly/numbertheory.html

il calcolo. Anche in questo caso dovrebbe essere abbastanza facile.

Una volta capito di cosa ha bisogno la procedura, possiamo passare all'algoritmo vero e proprio.

Analizza bene l'esempio precedente, ci sono alcuni passi che vengono ripetuti più volte. Riesci a scrivere in modo preciso (rigoroso) e generale (non dipendenti dal particolare numero scelto) questi passi e le operazioni da compiere?

In particolare, il cuore del processo è la moltiplicazione del numero attuale per il prodotto dei precedenti: questa è l'istruzione che va ripetuta più volte.

Una volta specificata la/le istruzioni, i passi, fondamentali, quelli che saranno eseguiti a ogni iterazione, specifica una condizione da soddisfare per interrompere il ciclo.

Il ciclo e le istruzioni che contiene sono il nocciolo dell'algoritmo.

Completa il lavoro precisando l'output della funzione, il risultato da restituire.

Una volta completata la procedura, prova a usarla con altri valori cercando di seguire in modo rigoroso le istruzioni che hai scritto (senza dare nulla per scontato) e controlla che ogni volta restituisca il risultato atteso.

Ora prova a usare la procedura fornendo come input zero oppure uno. C'è qualche problema? Se è così, prova a correggere la procedura (magari aggiungendo dei controlli all'inizio) per gestire questi casi particolari.

In pratica, si tratta di *implementare* meglio la definizione di fattoriale trattando a parte, magari all'inizio, prima del ciclo di cui abbiamo parlato qui sopra, alcuni casi specifici, che hanno necessità di un trattamento differente.

Se sei riuscito a modificare lo pseudo-codice che descrive il procedimento, consideriamo un'altra possibilità.

Prova a calcolare il fattoriale, usando la procedura che abbiamo appena scritto, di numeri negativi. Funziona correttamente? Probabilmente è necessario aggiungere un ulteriore controllo, all'inizio, perchè il fattoriale dei numeri negativi non è definito (o almeno, non lo è per i nostri scopi). Allo stesso modo, dovremmo controllare che il numero sia intero.

Si tratta di avere pazienza, spesso una buona procedura ha bisogno di un po' di tempo per essere affinata, considerando casi particolari che, per quanto poco frequenti, potrebbero indurre in errore. Vorremmo consegnare a un ipotetico utente uno pseudo-codice che sia in grado, da solo, di prevedere eventuali errori e gestire queste situazioni un po' marginali.

2.4. Scomporre in fattori

L'ultima funzione indispensabile per l'approssimazione dei numeri trascendenti, l'ultimo *mattoncino* che ci serve, è la scomposizione in fattori primi. Ancora una volta, puoi passare direttamente al paragrafo successivo, xx Metodo di Taylor, che trovi a pagina yy, se i metodi e i procedimenti ti sono chiari.

Ma cosa significa scomporre in fattori primi? Significa trovare tutti i fattori primi $p_1, p_2, \ldots p_k$ che, moltiplicati tra loro danno il numero naturale n specificato.

In termini più formali:

$$n = p_1 \cdot p_2 \cdot \ldots \cdot p_k = \prod_{i=1}^{k} p_i$$

Questa funzione deve restituire un output solo se l'input è un numero intero diverso da 0. Mentre se l'input è un numero negativo, l'output deve essere il numero stesso, cambiato di segno.

Useremo un piccolo trucco per stabilire se un numero n è divisibile per un numero p: se il resto della divisione di p per n è zero allora p divide n (o, in altre parole, n è divisibile per p), in caso contrario questo non è vero. Facciamo questo perchè moltissimi linguaggi di programmazione e generatori di codice hanno un operatore o una funzione che restituisce il resto della divisione p/n e quindi è molto semplice implementare il controllo in questo modo.

Cominciamo con un esempio

Il Toolbox: http://researchinaction.it/wp-content/uploads/2018/11/00-Toolbox.pdf suggerisce alcuni metodi per il calcolo dell'errore (cfr. Calcolo dell'errore a pagina 5).

Proviamo a scomporre in fattori primi un numero *facile* come 6.

» Partiamo da uno e cerchiamo il numero primo successivo, è il numero 2. Ora proviamo a dividere 6 per 2, il resto della divisione è 0? Allora abbiamo trovato il primo fattore! Infatti, per quanto detto sopra, se il resto è nullo abbiamo trovato un divisore. Mettiamo da parte questo numero: è il primo fattore primo di sei.
» Dividiamo 6 per 2 ottenendo 3. Questo risultato è ancora divisibile per 2? No, perchè il resto non è zero, questo vuol dire che sei non è ulteriormente divisibile per due.
» Quindi continuiamo con il numero primo successivo a due, ossia 3. Il risultato della divisione precedente (3) è divisibile per 3? La risposta è ancora positiva e quindi 3 è anch'esso un fattore di 6. Mettiamo da parte anche questo fattore.
» Dividiamo ancora il numero attuale (3) per il fattore trovato (sempre 3), visto che otteniamo uno come risultato, siamo giunti alla fine della procedura! Non ci sono altri fattori primi di sei.

In conclusione otteniamo la lista *(2, 3)* che contiene tutti i fattori primi di sei: abbiamo la nostra scomposizione.

La procedura

Scriviamo ora, concretamente la procedura in generale.

Ancora una volta, per iniziare, precisa l'input della procedura, cioè tutto quello che serve perchè la procedura esegua il calcolo.

Una volta precisato l'input, procedi con l'algoritmo.

Questa possibilità di riusare funzioni e procedure già realizzate (o realizzate da altri) è uno dei più grandi vantaggi della programmazione, del coding. Non sempre (quasi mai) si ha bisogno di scrivere tutto il codice (o lo pseudo-codice) che ci serve, a volte lo abbiamo già fatto risolvendo un problema diverso, a volte lo ha fatto qualcun altro. Come già detto, le procedure più semplici sono *mattoncini* per costruire procedure più complesse.

Analizza bene l'esempio precedente, ci sono alcuni passi che vengono ripetuti più volte. Riesci a scrivere in modo preciso (rigoroso) e generale (non dipendenti dal particolare numero scelto) questi passi e le operazioni da compiere?

In particolare , questa volta, devi focalizzarti su quale sia il quoziente che ti fa capire che la procedura è giunta al termine. Per capirlo, ti conviene ripetere la scomposizioni in fattori, con altri numeri *facili*. Prova con 7,8,9,12, ... Tieni conto che ogni volta che nell'esempio abbiamo trovato un divisore, un fattore primo, abbiamo diviso il numero originale proprio per quel fattore e abbiamo testato, con i primi successivi, non tanto il numero che ci hanno dato quanto il risultato della divisione. È questo il nocciolo della questione.

Tieni conto che diamo per scontato di avere a disposizione una funzione, che chiameremo PrimoSuccessivo, che, fornendo un numero intero, restituisce il primo immediatamente successivo:

$$\text{PrimoSuccessivo}(x)$$

Si tratta della procedura/funzione formalizzata nel paragrafo 2.4 La prima procedura: il primo successivo p a pagina 8. Se non hai *saltato* quel paragrafo poco male, usa comunque la funzione *PrimoSuccessivo* senza preoccuparti dei dettagli, in qualche modo questa restituirà proprio il valore che ti serve.

Come sempre, se sei riuscito a individuare le istruzioni essenziali, quelle che vengono ripetute più volte, *racchiudile* in un ciclo e cerca di specificare bene la condizione che, se verificata, porrà termine al ciclo.

Ora non rimane che fissare l'attenzione sul risultato. Tieni conto che, questa volta, non vogliamo un numero quanto piuttosto una *lista* di numeri.

Precisa l'output della procedura, specifica in modo scrupoloso cosa deve essere restituito a un ipotetico utente come risultato del procedimento.

Ora che hai completato la procedura, prova a usarla con valori diversi. Come al solito cerca di essere severo: non dare niente per scontato ed esegui esattamente le istruzioni che hai formalizzato. Se necessario modifica e correggi, affina, la procedura in modo che funzioni correttamente. Funziona? Se sì, abbiamo l'algoritmo.

Prova a usare la procedura fornendo come input zero o un numero non intero. C'è qualche problema? Se è così, prova a correggere la procedura (magari aggiungendo dei controlli all'inizio, in modo simile a quanto già fatto in precedenza) per gestire questi casi particolari.

Se sei riuscito a aggiustare lo pseudo-codice che descriva il procedimento, consideriamo un'altra possibilità: se l'input è un numero negativo?

Prova a scomporre in fattori, usando la procedura che abbiamo appena scritto, un numero negativo. Funziona correttamente?

Probabilmente è necessario aggiungere un ulteriore controllo, all'inizio. Infatti, bisogna procedere trattando la scomposizione di un numero negativo come quella di un numero positivo. Solo alla fine, quando si moltiplicano gli output ottenuti per provare che siano realmente in fattori che compongono l'input, moltiplicheremo il prodotto finale per -1.

2.5. Il metodo di Taylor per approssimare il numero di Eulero

La formula di Taylor permette di approssimare una funzione $f(x)$ in un intorno di un punto x_0 mediante un polinomio costruito con i valori delle derivate successive della funzione:

$$f(x) = f(x_0) + f'(x_0)\cdot(x - x_0) + f''(x_0)\cdot\frac{(x - x_0)^2}{2!} + \ldots + f^{(n)}(x_0)\cdot\frac{(x - x_0)^n}{n!}$$

o meglio

$$f(x) = \sum_{i=0}^{n} f^{(i)}(x_0)\cdot\frac{(x - x_0)^i}{i!}$$

approssimazione a meno di un errore che non trattiamo per non appesantire troppo la trattazione.

Qui useremo la formula di Taylor per approssimare la funzione esponenziale e^x che, calcolata per x = 1, ci darà un'approssimazione del numero di Eulero $e = 2.71828182846$ ...

Infatti, sviluppando in serie di Taylor la funzione esponenziale si ottiene:

$$e^x \approx 1 + x + \frac{x^2}{2!} + \frac{x^3}{3!} + \ldots + \frac{x^k}{k!}$$

Se calcoliamo questo sviluppo in uno (sostituiamo uno alla x nell'espressione ottenuta) abbiamo:

$$e^1 \approx 1 + 1 + \frac{1^2}{2!} + \frac{1^3}{3!} + \ldots + \frac{1^k}{k!} = 1 + 1 + \frac{1}{2!} + \frac{1}{3!} + \ldots + \frac{1}{k!}$$

che sembra proprio una formula interessante, possiamo approssimare il numero di Eulero sommando via via il reciproco del fattoriale di un numero intero a partire da zero fino a ... quanto vogliamo, e più termini aggiungeremo, migliore sarà l'approssimazione che otterremo. Ricordiamo, per chiarezza, che il fattoriale di zero e di uno è uno.

Un esempio

Per esempio, calcolando la formula ottenuta per $k = 5$ si ha:

$$e^1 \approx 1 + 1 + \frac{1}{2} + \frac{1}{6} + \frac{1}{24} + \frac{1}{120} = 2.716666666666667$$

che ci da un'approssimazione corretta fino alla seconda cifra decimale!

In pratica si procede in questo modo:

» partiamo da uno, in un certo senso è la nostra prima approssimazione
» aggiungiamo all'approssimazione attuale (1) il reciproco del fattoriale di uno ottenendo due
» aggiungiamo ancora il recproco del fattoriale di due (pari a un mezzo)
» sommiano poi il reciproco del fattoriale di tre (pari a un sesto)
» ... e così via
» fino ad arrivare al reciproco del fattoriale di cinque che era il limite che si eravamo imposti.

Il risultato finale, l'ultima somma calcolata, è l'approssimazione cercata.

È bene notare che abbiamo già una funzione che ci consente di calcolare il fattoriale di un numero intero, l'abbiamo formalizzata, sviluppata, nel paragrafo 2.3 fattoriale a pagina xx e che useremo così com'è, senza preoccuparci dei dettagli.

Ora procediamo provando a scrivere, a formalizzare, una procedura per approssimare il numero di Eulero con questa formula.

Come di consueto, inizia precisando l'input della funzione. Cosa è assolutamente necessario sapere per usare la formula?

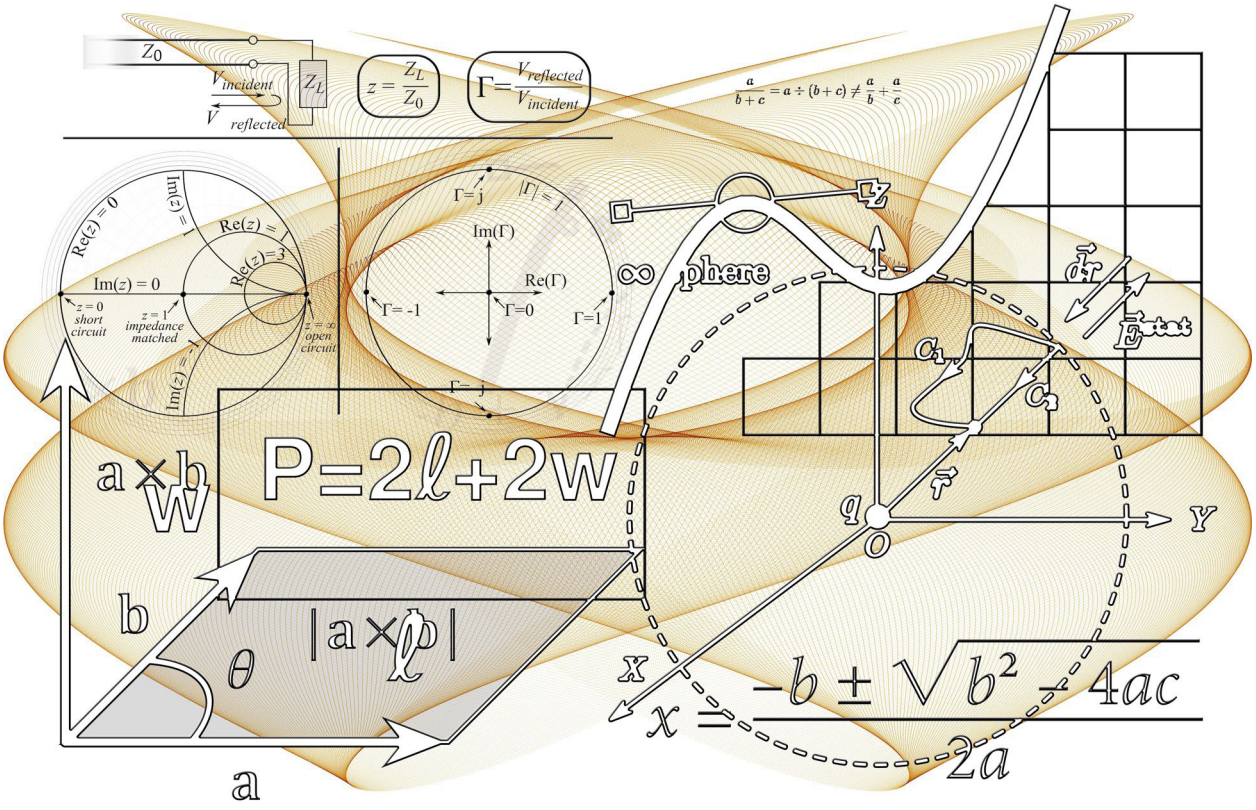
La formula è già lì bella e pronta ma può essere usata aggiungendo il numero di termini che si vuole. Probabilmente l'input, questa volta, deve riguardare proprio il numero di termini da usare.

Ora, come già fatto spesso in questo laboratorio, si tratta di specificare bene i passi, le istruzioni, che sono ripetute più volte, ciclicamente.

Ricordando l'esempio che abbiamo appena usato, si tratta, a ogni iterazione, di sommare un nuovo termine a un valore (a una somma) calcolato in precedenza. Probabilmente sarà necessario aggiungere un'istruzione, una *inizializzazione*, che assegni un valore iniziale a questa somma che poi modificheremo a ogni ciclo.

Infine, precisa la condizione che deve essere rispettata per concludere il ciclo. Se hai dubbi, analizza attentamente l'esempio che abbiamo fatto. Specifica quale deve essere il valore restituito.

A questo punto la procedura dovrebbe essere completa. Prova a *testarla* usando inizialmente un numero di iterazioni piccolo (due, tre, ...). Controlla che il comportamento sia proprio quello atteso, sia ancora una volta severo e non dare niente per scontato: una buona procedura non ha nulla di implicito ma *tutto avviene*, per così dire, *alla luce del Sole*.



Controlla come sempre che non ci siano problemi per i casi particolari. Per esempio, cosa succede se il numero di iterazioni è zero oppure negativo? Aggiungi eventualmente dei controlli per gestire queste situazioni.

Se tutto funziona, devi essere orgogliosa di te stessa, hai fatto davvero un buon lavoro!

2.6. Metodo di Newton per approssimare il numero di Eulero

Il metodo di Newton è un meccanismo utilizzato per determinare una soluzione approssimata di un'equazione del tipo $f(x) = 0$, a patto che la funzione $f(x)$ sia derivabile fino almeno al secondo ordine in un intervallo $[a, b]$ e che la derivata seconda, nell'intervallo considerato, sia di segno costante. Partendo da uno degli estremi dell'intervallo $[a, b]$ l'algoritmo procede avvicinandosi via via alla soluzione dell'equazione determinando, a ogni *step*, il punto successivo come intersezione della retta tangente alla curva nel punto precedente con l'asse delle ascisse. In altre parole, se x_0 è un'approssimazione della soluzione di $f(x) = 0$, una migliore approssimazione x1 si ottiene calcolando:

$$x_1 = x_0 - \frac{f(x_0)}{f'(x_0)}$$

Noi siamo interessati all'equazione:

$$\ln(x) - 1 = 0$$

che ha per soluzione proprio $x = e$! In realtà, come vedremo nel seguito, si potrà usare questo algoritmo, con delle piccole e semplici accortezze, anche per approssimare numeri irrazionali (come la radice quadrata di 2) o altri trascendenti come la sezione aurea φ.

Esula un po' dagli scopi di questo laboratorio, ma è comunque necessario precisare che il punto iniziale, la prima approssimazione della soluzione cercata, è $x_0 = a$ se $f(a)$ e la derivata seconda $f''(x)$ (che, ricordiamo, deve avere segno costante) hanno lo stesso segno, $x_0 = b$ se $f(a)$ e la derivata seconda $f''(x)$ hanno segno opposto.

In questo caso specifico la formula di Newton per determinare l'approssimazione successiva diventa, semplicemente:

$$x_1 = x_0 - \frac{f(x_0)}{f'(x_0)} = x_0 - \frac{\ln(x_0) - 1}{1/x_0} = x_0 - (\ln(x_0) - 1) \cdot x_0$$

Ripetendo iterativamente il calcolo si ottengono valori via via più vicini a e.

Un esempio

Scegliamo un intervallo che contenga il numero di Eulero, per esempio $[2, 3]$ e, quindi, poniamo $x_0 = 2$. Calcoliamo l'approssimazione successiva:

$$x_1 = 2 - (\ln(2) - 1) \cdot 2 \approx 2.6137056...$$

Se ora poniamo $x_0 = x_1$ e ripetiamo il calcolo si ha:

$$x_1 = 2.6137056 - (\ln(2.6137056) - 1) \cdot 2.6137056 \approx 2.7162439...$$

Come si vede abbiamo già ottenuto, dopo solo due iterazioni, due cifre decimali corrette!

La procedura

A questo punto, come scrivere la procedura? Per iniziare dobbiamo fornire alla nostra funzione un valore iniziale per x_0, possiamo partire dallo stesso valore scelto nell'esempio precedente.

Come di consueto, inizia precisando l'input della funzione. Cosa è assolutamente necessario sapere per usare la formula?

Come nel caso precedente la formula è già lì bella e pronta ma può essere usata aggiungendo il numero di termini che si vuole, quindi l'input, questa volta, deve riguardare il numero di termini da usare, le volte che vogliamo ripetere il calcolo, ma anche il valore di partenza.

Ora si tratta di specificare bene i passi, le istruzioni, che sono ripetute più volte, ciclicamente. In questo caso il calcolo del valore successivo, di una migliore approssimazione.

Immagina di generalizzare l'esempio precedente, c'è una formula che viene calcolata a ogni passo ma con valori via via diversi: è proprio questo calcolo che costituirà il cuore della procedura!

Infine, precisa la condizione che deve essere rispettata per concludere il ciclo. Se hai dubbi, analizza attentamente l'esempio che abbiamo fatto. Specifica quale deve essere il valore restituito.

Prova ora a seguire la procedura, per un piccolo numero di iterazioni, con carta, penna e calcolatrice e controlla che tutto funzioni in modo corretto. Questo metodo, come abbiamo anche visto nell'esempio, già con pochi cicli dovrebbe restituire qualche cifra decimale corretta.

2.7. Minimo scostamento

L'ultimo algoritmo di approssimazione di cui tratteremo in questo fascicolo è il *Minimo scostamento*, che può essere utilizzato per approssimare un numero reale come prodotto di razionali, ognuno dei quali ha per numeratore e denominatore due primi successivi. Il minimo scostamento può essere usato per approssimare un numero reale qualsiasi, che sia irrazionale, trascendente o altro ancora.

Il nome deriva dal fatto che l'approssimazione è modificata a ogni iterazione moltiplicando per una frazione, un rapporto di due primi successivi, in modo da ridurre la differenza tra l'attuale valore dell'approssimazione e il numero da approssimare.

Un esempio

Spieghiamo meglio con un esempio, per approssimare π si procede in questo modo:

» partiamo da uno, la prima approssimazione di π
» i due primi successivi a uno sono 2 e 3, se l'approssimazione attuale è minore di π la moltiplichiamo $3/2$, altrimenti (se è maggiore) per $2/3$
» ripetiamo il procedimento: i due primi successivi a tre sono 5 e 7, se l'approssimazione attuale è minore di π la moltiplichiamo $7/5$, altrimenti per $5/7$
» ... e così via fino al numero di iterazioni desiderato.

Ovviamente, maggiore è il numero di iterazioni, migliore sarà l'approssimazione. Nel caso di π abbiamo, con quattro iterazioni:

$$\pi \approx 1 \cdot \frac{3}{2} \cdot \frac{7}{5} \cdot \frac{13}{11} \cdot \frac{19}{17} = 1.232798573975045$$

La procedura

A questo punto, come sempre, avremo bisogno di un input da fornire alla funzione per farla lavorare.

Precisa le informazioni di cui hai bisogno per iniziare e se qualche variabile deve essere

inizializzata prima del ciclo principale della procedura.

Come altre volte in questo laboratorio, di sicuro ci sarà bisogno di fissare il numero di iterazioni ma, in questo caso, visto che il metodo può essere usato su un numero qualsiasi, si deve pensare anche a questo.

Concentrati ora sulle istruzioni principali, quelle che vanno ripetute a ogni iterazione. Cerca di formalizzare queste istruzioni, cerca di comprendere come modificare via via l'approssimazione.

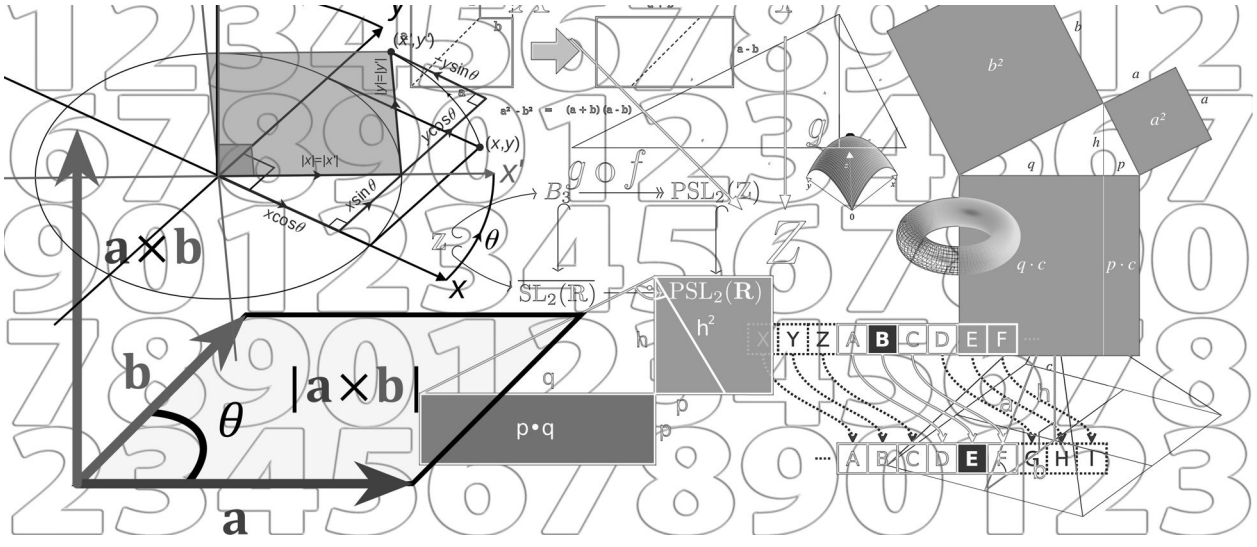
A ogni iterazione si moltiplica l'approssimazione per il rapporto dei due numeri primi successivi in modo da ridurre la differenza con il numero che cerchiamo di approssimare ed è proprio questa approssimazione che va modificata ogni volta. Un suggerimento: una volta *abbozzata* la funzione, presta attenzione al valore dell'approssimazione, le istruzioni che hai scritto devono aggiornare questo valore a ogni iterazione.

Tieni conto che, nel corso del laboratorio, abbiamo già sviluppato una funzione che ci restituisce il primo successivo a un numero dato: *PrimoSuccessivo(x)*, che tu abbia o no svolto quella parte del laboratorio, hai comunque la possibilità di usare la funzione corrispondente (cfr. 2.2 La prima procedura: il primo successivo p a pagina 8).

Ora fissa l'attenzione sulla condizione da imporre per fermare il ciclo e sulle informazioni, sul valore da restituire, all'uscita dalla procedura.

Metti alla prova la procedura su numeri reali facili, non necessariamente irrazionali o trascendenti, allo scopo di controllare il funzionamento della procedura.

E con questo abbiamo finito. Se ci hai seguito fino a qui, dobbiamo farti i complimenti: era tutt'altro che facile!



3. Soluzioni

Bene, se sei arrivato fino a questo punto, significa che in te c'è un vero matematico, capace di risolvere ogni problema, usando la logica e il duro lavoro! Vediamo un po' se le procedure che hai scritto sono simili a quelle che abbiamo svilupppato noi.

3.1. La prima procedura: il primo successivo p

La procedura

Precisa l'input della procedura, in altre parole tutto quello che serve perchè la procedura esegua il calcolo. Per questo caso particolare, dovrebbe essere abbastanza facile.

Come detto, è facile: ci serve solamente un numero intero scelto da noi da cui partire per la nostra ricerca, quindi, in termini formali, la prima istruzione dovrebbe essere questa:

```
PrimoSuccessivo(x)
      i = x + 1
```

dove *PrimoSuccessivo* è il nome della funzione e x è il numero proprio da cui partire. La seconda riga, invece, assegna alla variabile i il numero da cui partire aumentato di uno, in altre parole si tratta del numero intero successivo a x: il primo di cui andremo a controllare la primalità.

Analizza bene l'esempio precedente, ci sono alcuni passi che vengono ripetuti più volte. Riesci a scrivere in modo preciso (rigoroso) e generale (non dipendenti dal particolare numero scelto) questi passi e le operazioni da compiere?

Dobbiamo ripetere l'incremento del contatore fino a quando non diventa un numero primo (ricordate l'esempio che abbiamo fatto a pagina 8?), quindi c'è una sola istruzione che va ripetuta a ogni ciclo:


```
      i = i+1
```

Specifica quali frasi, istruzioni, devono essere ripetute più volte e indica chiaramente una condizione da soddisfare perchè il ciclo (composto dalle istruzioni ripetute) si interrompa.

Questo incremento va ripetuto fino a quando non arriviamo ad avere un numero primo:

```
ripeti fino a che i non diventa primo
      i = i+1
```

Specifica l'output dell'algoritmo, il risultato finale del processo.

Questo è facile, dobbiamo restituire il valore attuale del contatore (la variabile i) che al termine del ciclo è diventato il numero primo che cercavamo.

Ora, ricomponiamo l'intera procedura aggiungendo anche l'istruzione per ritornare il valore!

```
PrimoSuccessivo(x)
        i = x + 1
        ripeti fino a che i non diventa primo
                i = i+1
        ritorna i
```

3.2. Fattoriale

Precisa l'input della procedura, cioè tutto quello che serve perchè la procedura esegua il calcolo. Anche in questo caso dovrebbe essere abbastanza semplice.

Anche questa volta precisare l'input (i parametri da fornire alla funzione) non è troppo difficile: ci serve solamente un numero intero, quello di cui calcolare il fattoriale, quindi, in termini formali, la prima istruzione dovrebbe essere questa:

```
Fattoriale (x)
        p = 1
```

La variabile p conterrà il fattoriale che vogliamo calcolare. È impostata a uno perchè in seguito sarà moltiplicata per gli interi successivi e se fosse zero ogni prodotto sarebbe nullo.

Analizza bene l'esempio precedente, ci sono alcuni passi che vengono ripetuti più volte. Riesci a scrivere in modo preciso (rigoroso) e generale (non dipendenti dal particolare numero scelto) questi passi e le operazioni da compiere?

Ci serve un contatore che partendo dal primo numero che potrebbe essere un fattore (2) sia incrementato a ogni ciclo, fino a raggiungere il numero di cui vogliamo calcolare il fattoriale. Possiamo partire da 2 e non da uno come affermato nella definizione, perché la moltiplicazione per uno è ininfluente. L'istruzione corrispondente dovrebbe avere allora questo aspetto:

```
        p = p * i
```

Una volta specificata la/le istruzioni, i passi, fondamentali, quelli che saranno eseguiti a ogni iterazione, specifica una condizione da soddisfare per interrompere il ciclo.

Il ciclo deve ripetere la moltiplicazione che abbiamo appena visto fino a raggiungere il numero che è stato passato come parametro:

```
        per i che va da 1 a x
                p = p * i
```

Completa il lavoro precisando l'output della funzione, il risultato da restituire.

Il valore da restituire è proprio il prodotto, contenuto nella variabile *p*:

```
        ritorna p
```

Mettendo insieme tutti i *pezzi*, abbiamo la nostra procedura che calcola il fattoriale! Una volta completata la procedura, prova a usarla con altri valori cercando di seguire in modo rigoroso le istruzioni che hai scritto (senza dare nulla per scontato) e controlla che il risultato sia corretto.

Ora prova a usare la procedura fornendo come input zero oppure uno. C'è qualche problema? Se è così, prova a correggere la procedura (magari aggiungendo dei controlli all'inizio) per gestire questi casi particolari.

Il valore da restituire è ovviamente il prodotto *p*. Purtroppo il ciclo, così come lo abbiamo scritto non funziona correttamente quando x è zero oppure uno, dobbiamo aggiungere un paio di con-

Sul blog Research in Action è possibile utilizzare la libreria sviluppata con Blockly nel corso del laboratorio: http://researchinaction.it/myblockly/numbertheory.html
Blockly è un generatore di codice open source sostenuto da Google: https://developers.google.com/blockly/.

trolli per gestire queste due situazioni particolari. Così come sono necessari controlli per evitare problemi quando x è un numero reale (non intero) o negativo. Nel complesso la funzione dovrebbbe avere questo aspetto:

```
Fattoriale (x)
        se x è zero allora restituisci 1
        se x è uno allora restituisci 1
        se x < 0 allora restituisci +∞
        se x è non è intero allora restituisci +∞
              p = 1
        per i che va da 2 a x
              p = p * i
        ritorna p
```

Con l'istruzione *per i che va da 1 a x* si intende che le istruzioni che seguono, che fanno parte del *blocco*, sono ripetute ogni volta aumentando il valore di i di uno a partire da due. Per cui, al primo giro la i vale 2, al secondo 3, … l'ultima volta che il ciclo è eseguito la i vale x.

3.3. Scomporre in fattori

Ancora una volta, per iniziare, precisa l'input della procedura, cioè tutto quello che serve perchè la procedura esegua il calcolo.

Allora, l'unica cosa che serve per cominciare, è fornire il numero che si vuole scomporre in fattori, per cui la prima riga della funzione dovrebbe avere più o meno questo aspetto:

```
ScomporreInFattori(n)
```

Analizza bene l'esempio precedente, ci sono alcuni passi che vengono ripetuti più volte. Riesci a scrivere in modo preciso (rigoroso) e generale (non dipendenti dal particolare numero scelto) questi passi e le operazioni da compiere?

Dobbiamo inserire un ciclo, un *loop*, che ripeta la divisione tra n e via via numeri primi successivi a partire da due (infatti, se partissimo da uno la divisione sarebbe inutile la funzione continuerebbe all'infinito a ripetere lo stesso calcolo). Ogni volta che n non sarà più divisibile per il primo che stiamo usando dovremo provare con il primo successivo a quello in uso.


```
l = crea una lista vuota
p = 2
ripeti fino a che n = 1
      se n è divisibile per p allora
            in l inserisci p all'ultimo posto
            n = n / p
      altrimenti
            p = PrimoSuccessivo(p)
```

Anticipiamo una rispota che troverete più avanti. La funzione deve restituire tutti i fattori del numero specificato e quindi, a ogni ciclo, se troviamo un fattore, lo aggiungiamo all'utlimo posto della lista.

Come sempre, se sei riuscito a individuare le istruzioni essenziali, quelle che vengono ripetute più volte, *racchiudile* in un ciclo e cerca di specificare bene la condizione che, se verificata, porrà termine al ciclo.

Il ciclo si conclude, ovviamente, quando il contatore arriva al numero da scomporre. Aggiungiamo un paio di controlli per gestire il caso in cui il numero da scomporre non sia un intero positivo.

```
ScomporreInFattori(n)
        l = crea una lista vuota
        se n = 0 oppure n non è intero allora
                ritorna l
        se n < 0 allora
                n = -1 * n
        p = 2
        ripeti fino a che n = 1
                se n è divisibile per p allora
                        in l inserisci p all'ultimo posto
                        n = n / p
                altrimenti
                        p = PrimoSuccessivo(p)

        ritorna l
```

Se alla funzione viene passato come parametro un numero che non è intero oppure nullo la funzione restituisce una lista vuota, proprio perchè la scomposizione è impossibile. Invece, il parametro è intero ma negativo si moltiplica si cambia di segno al parametro e si procede trattando la scomposizione di un numero negativo come quella di un numero positivo.

Sul blog Research in Action è possibile utilizzare la libreria sviluppata con Blockly nel corso del laboratorio: http://researchinaction.it/myblockly/numbertheory.html
Blockly è un generatore di codice open source sostenuto da Google: https://developers.google.com/blockly/.

3.4. Il metodo di Taylor per approssimare il numero di Eulero

Come di consueto, inizia precisando l'input della funzione. Cosa è assolutamente necessario sapere per usare la formula?

Abbiamo bisogno solo del numero di iterazioni, quindi:

```
Taylor_e(iterazioni)
        p = 1
```

dove *Taylor_e* è la nostra funzione e *iterazioni* è il nostro input, ossia il numero di volte che sarà ripetuto il ciclo principale. La seconda istruzione inizializza la variabile al primo termine della serie che ci permetterà di approssimare il numero di Eulero.

Ora, come già fatto spesso in questo laboratorio, si tratta di specificare bene i passi, le istruzioni, che sono ripetute più volte, ciclicamente.

Si tratta di aggiungere a ogni ciclo all'approssimazione attuale il termine successivo della successione. In altre parole, a ogni ciclo aggiungiamo il reciproco del fattoriale di un contatore che inizialmente vale uno e a ogni ietrazione è incrementato di uno.

```
        p = p + 1/ Fattoriale(l)
```

In pratica, alla prima iterazione somma il reciproco del fattoriale di uno, alla seconda il reciproco del fattoriale di due, … e la somma viene conservata nella variabile p. In questo modo, a ogni giro, si ottiene di aggiungere il reciproco alla somma dei precedenti. Notare che abbiamo fatto uso della funzione *Fattoriale(n)* definita in precedenza.

Infine, precisa la condizione che deve essere rispettata per concludere il ciclo. Se hai dubbi, analizza attentamente l'esempio che abbiamo fatto (cfr. 2.6 Metodo di Newton per approssimare il numero di Eulero a pagina 15). Specifica quale deve essere il valore

Sul blog Research in Action è possibile utilizzare la libreria sviluppata con Blockly nel corso del laboratorio: http://researchinaction.it/myblockly/numbertheory.html
Blockly è un generatore di codice open source sostenuto da Google: https://developers.google.com/blockly/.

restituito.

Il ciclo deve essere ripetuto, come già detto, fino a quando il contatore non raggiunge il valore del parametro fornito alla funzione. Il valore da restituire è proprio la somma dei reciproci dei fattoriali contenuta nella variabile *p*. A questo punto la funzione è completa.

```
Taylor_e(iterazioni)
      se iterazioni < 1 allora ritorna -inf
      iterazioni = round(iterazioni)
      p = 1
      per i che va da 1 a iterazioni
            p = p + 1/ Fattoriale(l)
      ritona p
```

Abbiamo anche aggiunto un paio di istruzioni all'inizio per gestire i casi in cui il numero di iterazioni fornito sia minore di uno o non intero.

3.5. Metodo di Newton per approssimare il numero di Eulero

Come di consueto, inizia precisando l'input della funzione. Cosa è assolutamente necessario sapere per usare la formula?

In questo caso l'input di questa funzione deve essere il numero di iterazioni. Infatti è necessario che il metodo di Newton, la sua formula, applicata all'equazione

$$\ln(x) - 1 = 0$$

venga ripetuta più e più volte, migliorando l'approssimazione a ogni passaggio. Più è grande il numero di iterazioni, migliore sarà il risultato. L'istruzione per l'input sarà quindi:

```
Newton_e(iterazioni)
      x_0 = 2
```

Abbiamo aggiunto, come prima istruzione, l'inizializzazione della variabile x_0 che sarà il punto di partenza del metodo.

Ora si tratta di specificare bene i passi, le istruzioni, che sono ripetute più volte, ciclicamente. In questo caso il calcolo del valore successivo, di una migliore approssimazione.

E' facile: la nostra funzione deve semplicemente ripetere la formula di Newton

$$x_0 = x_0 - (\ln(x_0) - 1) \cdot x_0$$

già vista (cfr. 2.6 Metodo di Newton per approssimare il numero di Eulero a pagina 15), quindi:

```
x_0 = x_0 - (ln(x_0) - 1) * x_0
```

Infine, precisa la condizione che deve essere rispettata per concludere il ciclo. Se hai dubbi, analizza attentamente l'esempio che abbiamo fatto. Specifica quale deve essere il valore restituito.

L'istruzione che calcola l'approssimazione deve essere ripetuta *iterazioni* volte, per cui possiamo usare una struttura, un blocco di codice già visto:

```
ripeti iterazioni volte
      x_0 = x_0 - (ln(x_0) - 1) * x_0
```

Il valore restituito deve essere necessariamente x_0, infatti, a ogni ciclo, il rislutato della formula di Newton è memorizzato, conservato, in questa variabile. In conclusione, la nostra funzione avrà l'aspetto che segue:

```
Newton_e(iterazioni)
      x_0 = 2
      ripeti iterazioni volte
            x_0 = x_0 - (ln(x_0) - 1) * x_0
      ritorna x_0
```

Sul blog Research in Action è possibile utilizzare la libreria sviluppata con Blockly nel corso del laboratorio: http://researchinaction.it/myblockly/numbertheory.html Blockly è un generatore di codice open source sostenuto da Google: https://developers.google.com/blockly/.

3.6. Minimo scostamento

Precisa le informazioni di cui hai bisogno per iniziare e se qualche variabile deve essere inizializzata prima del ciclo principale della procedura.

Abbiamo, ovviamente, la necessità di conoscere il numero da approssimare ma anche il numero di iterazioni dopo il quale fermare la ricerca dell'approssimazione per cui::

```
MinimoScostamento(ripetizioni, trascendente)
      p2 = 1
      t = 1
```

Abbiamo anche aggiunto l'inizializzazione di un paio di variabili: *p2* è il secondo dei numeri primi successivi mentre *t* è l'approssimazione in prima istanza.

In realtà la procedura MinimoScostamento che potete trovare nella libreria dedicata usa una procedura chiamata MinimoScostamentoLista che invece di calcolare l'approssimazione come prodotto di razionali (ognuno dei quali è un quoziente di primi), salva questi fattori primi in una lista (da cui il nome). La funzione MinimoScostamento usa poi la lista per calcolare l'approssimazione moltiplicando tutti i primi con l'esponente corretto a seconda che siano a numeratore o a denominatore. Il motivo è che la successione di fattori primi (con esponente uno o meno uno) è usata nella derivata aritmetica per cui si è preferito scrivere una procedura più generale, che potesse essere usata in entrambi i casi.

Concentrati ora sulle istruzioni principali, quelle che vanno ripetute a ogni iterazione. Cerca di formalizzare queste istruzioni, cerca di comprendere come modificare via via l'approssimazione.

In pratica, a ogni ciclo, dobbiamo determinare i successivi numeri primi e moltiplicare l'approssimazione attuale per il quoziente di questi due numeri primi, quoziente costruito in modo da ridurre lo scostamento dal numero da approssimare: se l'approssimazione attuale è più piccola del numero trascendente il quoziente è maggiore di uno, in caso contrario è minore di uno.

```
p1 = PrimoSuccessivo(p2)
p2 = PrimoSuccessivo(p1)
se t < trascendente allora
      t = t * p2 / p1
altrimenti
      t = t * p1 / p2
```

Come si vede usiamo la funzione PrimoSuccessivo che abbiamo elaborato in precedenza (cfr. 2.2 La prima procedura: il primo successivo p a pagina 8).

Ora fissa l'attenzione sulla condizione da imporre per fermare il ciclo e sulle informazioni, sul valore da restituire, all'uscita dalla procedura.

Si tratta di ripetere il blocco di istruzioni precedente per il numero di iterazioni specificato:

```
ripeti ripetizioni volte
      ...
ritorna t
```

A te il compito di assemblare le varie parti per costruire la procedura completa.

4. Ultimo step!

Sul blog Research in Action è possibile utilizzare la libreria sviluppata con Blockly nel corso del laboratorio: http://researchinaction.it/myblockly/numbertheory.html
Blockly è un generatore di codice open source sostenuto da Google: https://developers.google.com/blockly/.

Come avrai letto, per il sommo π ci ha aiutato principalmente *Mr. Frederick Taylor*, regalandoci la sua formula che abbiamo usato per determinare un'approssimazione di questo numero, nella libreria che abbiamo realizzato si chiamano *Taylor_atan_pi* e *Taylor_atan_1-2_1-3_pi*. Terza ed ultima funzione per π, è il *MinimoScostamento*.

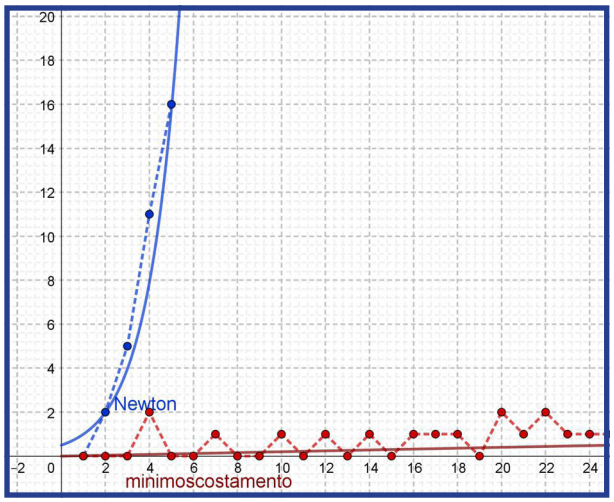
Passando al numero di Eulero *e*, anche Newton ha dato il suo contributo con *Newton_e* comunque affiancato da *Taylor_e* (sempre funzioni codificate grazia a Blockly). Infine, per avvicinarci alla radice quadrata di due, abbiamo costruito e utilizzato le funzioni *Newton_sqrt2* e ancora il *MinimoScostamento*.

A questo punto, visto che potevano sfruttare la potenza di calcolo di un computer, abbiamo messo alla prova i vari metodi di approssimazione per stimare la rapidità di convergenza (la velocità con cui fornivano una buona approssimazione). Abbiamo rappresentato graficamente i risultati e approssimati (ancora questo verbo) con una curva per dedurre quale metodo sia il più efficace!

Ovviamente, visto che la libreria è disponibile sul nostro *blog*, puoi provare da solo i vari metodi. Un suggerimento per iniziare: ti consigliamo di rappresentare sulle ascisse il numero di iterazioni e sulle ordinate le cifre decimali corrette, così come abbiamo fatto noi qui di seguito.

Per esempio, utilizzando il metodo di Newton per approssimare la radice quadrata di due, scegliendo una sola iterazione, non ci sono cifre decimali corrette. Quindi, il primo punto della curva che rappresenta l'efficienza del nostro metodo è, in questo esempio, *(1,0)*.

4.1. Radice quadrata di due



Come è evidente dal grafico, il metodo di Newton raggiunge un'ottima approssimazione dopo appena cinque iterazioni. Per questo, viene rappresentato da una funzione esponenziale *molto ripida*, con uno scarto quadratico pari a *2.35*;

D'altra parte vediamo che impiegando il minimo scostamento per trovare un'approssimazione decente dobbiamo scegliere un numero elevato di iterazioni. I dati raccolti possono essere rappresentati in questo caso da una funzione lineare con un coefficiente angolare molto basso, circa *0.02*, con una stima dell'errore (scarto quadratico) di *0.55*.

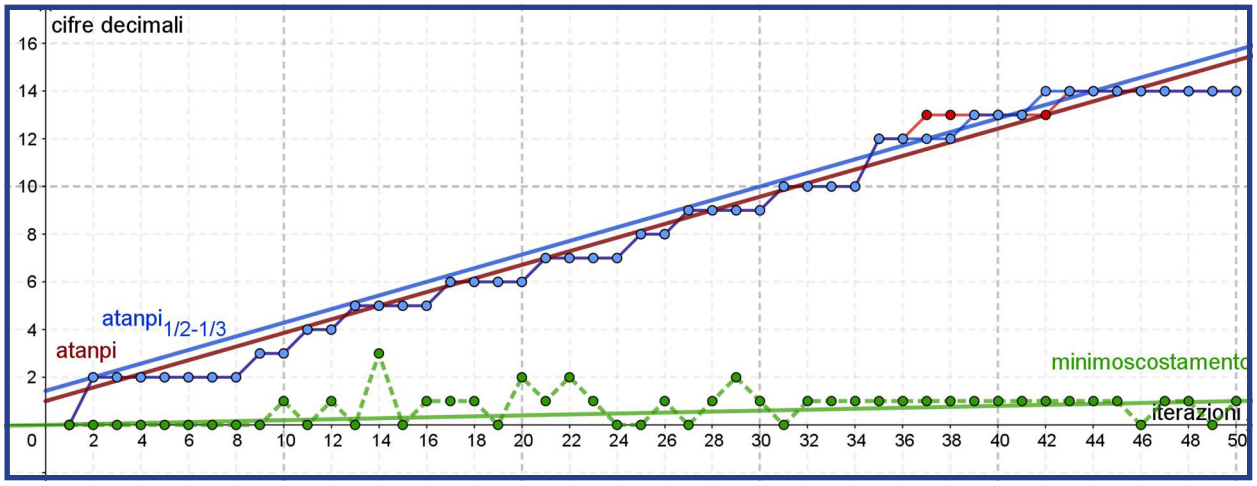
Le stime della convergenza delle approssimazioni di radice quadrata di due sono mostrate nella figura qui sopra: in colore blu il metodo di Newton e in colore rosso il *minimo scostamento*. Basta un'occhiata per notare la differenza.

Il Toolbox: http://researchinaction.it/wp-content/uploads/2018/11/00-Toolbox.pdf suggerisce alcuni metodi per l'approssimazione di dati sperimentali (cfr. Approssimazione mediante polinomi a pagina 13).

4.2. Pi greco

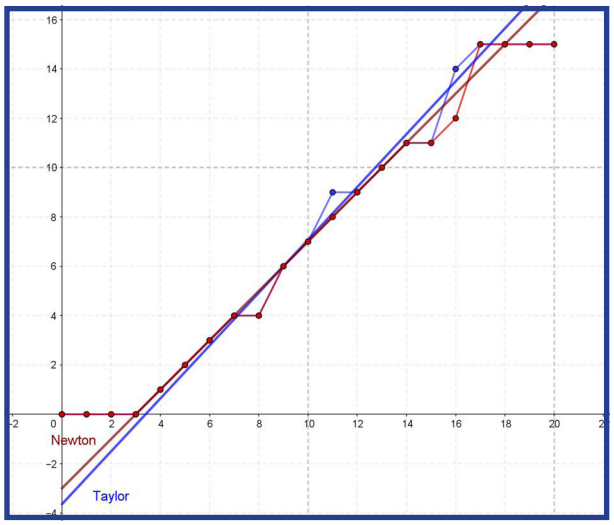
Stesso lavoro per π. Questa volta abbiamo confrontato tre metodi di approssimazione: due si basano sullo sviluppo in serie di Taylor dell'arcotangente mentre il terzo è di nuovo il metodo del *Minimo scostamento*.

I primi due metodi calcolano lo sviluppo dell'arcotangente in uno oppure sommano l'arcotangente di *1/2* e di *1/3*, si tratta di due varianti dello stesso metodo che sostanzialmente si equivalgono



e sono assimilabili a una crescita lineare, con coefficiente angolare uguale e quota leggermente diversa. Anche in questo caso il metodo del minimo scostamento è di gran lunga il peggiore!

Il grafico è mostrato qui sopra. I dati dell'approssimazione con l'arcotangente di uno sono in colore blu, la somma dell'arcotangente di *1/2* e *1/3* in rosso, e il minimo scostamento in verde. Le stime sono fornite con un errore (scarto quadratico medio) rispettivamente di *0.41*, *0.71* e *0.45*.



4.3. Il numero di Eulero

Concludiamo con la rappresentazione grafica della stima della convergenza dei metodi applicati al numero di Eulero *e*. Questa volta abbiamo tralasciato il metodo del Minimo scostamento: appare chiaro che converge molto molto lentamente.

Nella figura che segue la stima per il metodo di Newton è in colore rosso e quelle del metodo di Taylor (sviluppo in serie della funzione esponenziale, successivamente calcolata in uno) in colore blu. La convergenza è pressapoco lineare, con le due rette che hanno un coefficiente angolare abbastanza simile.

5. Esercizi

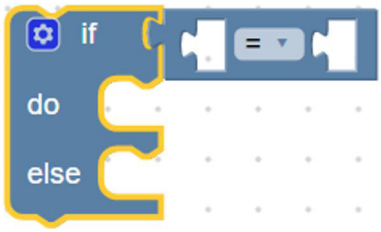
5.1. Codifica con Blockly

Costruire, tradurre, le procedure scritte nel linguaggio quasi-formale che abbiammo usato in codice eseguibile utilizzando Blockly. Abbiamo cercato quanto più possibile di formalizzare gli algoritmi in modo da far somigliare pezzetti del nostro *pseudo-codice* ai blocchi di Blockly. Per esempio, le istruzioni

```
se ... condizione ... allora
      ... istruzioni ...
altrimenti
      ... altre istruzioni ...
```

corrispondono quasi esattamente al blocco *IfElse* del generatore di codice di Google (mostrato nella figura che segue) quindi il lavoro dovrebbe essere facilitato.

Sul blog Research in Action è possibile utilizzare la libreria sviluppata con Blockly nel corso del laboratorio: http://researchinaction.it/myblockly/numbertheory.html
Blockly è un generatore di codice open source sostenuto da Google: https://developers.google.com/blockly/.



5.2. Estensione ad altri trascendenti

Pi greco, una prima approssimazione

Si può applicare un metodo simile a quello descritto in 2.5 Il metodo di Taylor per approssimare il numero di Eulero a pagina 13 per determinare un'approssimazione del rapporto tra lunghezza di una circonferenza e diametro, il numero π. Infatti è possibile sviluppare in serie di Taylor la funzione arcotangente:

$$\operatorname{atan}(x) \approx x - \frac{x^3}{3} + \frac{x^5}{5} - \frac{x^7}{7} + \frac{x^9}{9} + \dots$$

e ricordando che

$$\operatorname{atan}(1) = \frac{\pi}{4}$$

abbiamo una formula per il nostro famigerato π:

$$\pi = 4 \cdot \operatorname{atan}(1) \approx 4 \cdot \left(1 - \frac{1}{3} + \frac{1}{5} - \frac{1}{7} + \frac{1}{9} + \dots\right)$$

ottenuta sostituendo uno alla x nello sviluppo dell'arcotangente in serie di taylor.

Cerca di scrivere una funzione, così come fatto nel corso del laboratorio, per determinare un'approssimazione di π usando la formula che abbiamo appena costruito.

5.3. Pi greco, un'approssimazione migliore

Possiamo fare ancora meglio! È possibile far vedere, con una elegante dimostrazione geometrica, che

$$\frac{\pi}{4} = \operatorname{atan}\left(\frac{1}{2}\right) + \operatorname{atan}\left(\frac{1}{2}\right)$$

per cui, usando ancora lo sviluppo in serire di Taylor dell'arcotangente che abbiamo visto in precedenza possiamo scrivere

$$\pi = 4 \cdot \left[1 - \frac{1}{3}\left(\frac{1}{2}\right)^3 + \frac{1}{5}\left(\frac{1}{2}\right)^5 - \frac{1}{7}\left(\frac{1}{2}\right)^7 + \frac{1}{59}\left(\frac{1}{2}\right)^9 + \ldots\right] +$$
$$+4 \cdot \left[1 - \frac{1}{3}\left(\frac{1}{3}\right)^3 + \frac{1}{5}\left(\frac{1}{3}\right)^5 - \frac{1}{7}\left(\frac{1}{3}\right)^7 + \frac{1}{59}\left(\frac{1}{2}\right)^9 + \ldots\right]$$

Prova a scrivere una funzione, abbastanza simile alla precedente, che calcoli un'approssimazione di π come somma dell'arcotangente di un mezzo e di un terzo (in altre parole, usando la formula che abbiamo scritto qui sopra).

5.4. Estensione ad altri irrazionali

Radice quadrata di due

In 2.6 Metodo di Newton per approssimare il numero di Eulero a pagina 15 abbiamo usato il metodo di Newton per determinare una soluzione approssimata di un'equazione algebrica e, quindi, ricavare un'approssimazione del numero di Eulero. Possiamo applicare lo stesso metodo a una qualsiasi equazione algebrica che abbia come soluzione un numero irrazionale per determinare un'approssimazione di questo numero. Per esempio l'equazione

$$x^2 - 2 = 0$$

ha come soluzione la radice quadrata di due. Per questa equazione la formula di Newton diventa

$$x_1 = x_0 - \frac{x_0^2 - 2}{2x_0}$$

Prova a scrivere una funzione per approssimare la radice quadrata di due usando il metodo di Newton.

Sezione aurea

La sezione aurea è un numero irrazionale pari a

$$\phi = \frac{1+\sqrt{5}}{2} \approx 1.6180339887\ldots$$

Scrivi una procedura per approssimare la radice quadrata di cinque usando il metodo di Newton (è praticamente uguale a quella costruita per la radice quadrata di 2) e fai in modo che restituisca un'approssimazione della seziona aurea.

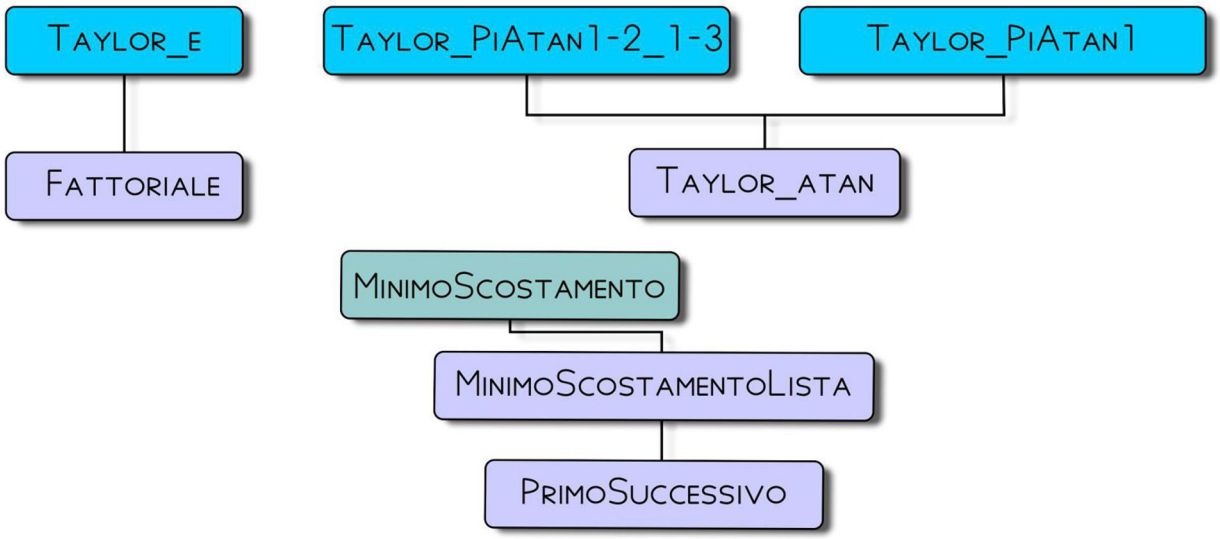
6. La libreria

Sul blog Research in Action è possibile utilizzare la libreria sviluppata con Blockly nel corso del laboratorio: http://researchinaction.it/myblockly/numbertheory.html
Blockly è un generatore di codice open source sostenuto da Google: https://developers.google.com/blockly/.

Contestualmente alla soluzione del problema, nel corso del laboratorio le funzioni realizzate sono state implementate mediante *Blockly*. In questo ultimo *capitoletto* è mostrata la composizione e la struttura di questa libreria.

6.1. Approssimazione di numeri irrazionali e trascendenti

La figura qui di seguito riporta le dipendenze tra le varie funzioni della libreria dedicate all'approssimazione dei numeri irrazionali e trascendenti: le funzioni più in alto nella figura usano le funzioni più in basso a cui sono collegate. Per esempio: *MinimoScostamento* usa *MinimoScostamentoLista* che a sua volta usa *PrimoSuccessivo*. Le funzioni che approssimano con il metodo di Newton non usano nessun altra funzione.



Approssimazione del numero di Eulero e tramite serie di Taylor

Approssimazione mediante lo sviluppo in serie di Taylor, in un intorno di zero, della funzione esponenziale, sviluppo calcolato poi per $x = 1$:

$$e^x \approx 1 + x + \frac{x^2}{2!} + \frac{x^3}{3!} + \ldots = \sum_{k=1}^{n} \frac{x^k}{k!}$$

da cui:

$$e^1 \approx 1 + 1 + \frac{1}{2!} + \frac{1}{3!} + \ldots = \sum_{k=1}^{n} \frac{1}{k!}$$

```
Taylor_e(iterazioni)
```

dove *iterazioni* è il numero di iterazioni usato nello sviluppo in serie. Restituisce un numero reale. Usa *Fattoriale(x)*.

Approssimazione del numero di Eulero e con il metodo di Newton

Il numero di Eulero e è approssimato ricercando una soluzione numerica dell'equazione:

$$\ln(x) - 1 = 0$$

Equazione usata per determinare un'approssimazione di *e* con il metodo di Newton, l'equazione

infatti ha come soluzione proprio il numero di Eulero.

```
Newton_e(iterazioni)
```

dove *iterazioni* è il numero di iterazioni usato nello sviluppo in serie. Restituisce un numero reale.

Approssimazione di pi greco tramite serie di Taylor

L'approssimazione è ottenuta sviluppando la funzione arcotangente in un intorno di zero e calcolando poi in un angolo (punto) opportuno.

$$\operatorname{atan}(x) \approx x - \frac{x^3}{3} + \frac{x^5}{5} - \frac{x^7}{7} + \frac{x^9}{9} + \ldots$$

```
Taylor_PiAtan1(iterazioni)
Taylor_PiAtan1-2_1-3(iterazioni)
```

dove *iterazioni* è il numero di iterazioni usato nello sviluppo in serie. Entrambe le funzioni restituiscono un numero reale.

La prima funzione calcola l'arcotangente in uno e moltiplica il risultato per quattro. La seconda utilizza la proprietà:

$$\frac{\pi}{4} = \operatorname{atan}\left(\frac{1}{2}\right) + \operatorname{atan}\left(\frac{1}{3}\right)$$

e quindi somma l'arcotangente di 1/2 e di 1/3 e moltiplica il risultato per quattro. Usa *Taylor_atan(iterazioni, x)*.

Approssimazione di radice di due con il metodo di Newton

L'approssimazione è calcolata applicando il metodo di Newton all'equazione:

$$x^2 - 2 = 0$$

```
Newton_r2(iterazioni)
```

dove iterazioni è il numero di iterazioni usato nello sviluppo in serie. Restituisce un numero reale.

Approssimazione della sezione aurea con il metodo di Newton

L'approssimazione è calcolata applicando il metodo di Newton all'equazione:

$$x^2 - 5 = 0$$

e l'approssimazione ottenuta è usata per calcolare la sezione aurea.

```
Newton_phi(iterazioni)
```

dove *iterazioni* è il numero di iterazioni usato nello sviluppo in serie. Restituisce un numero reale.

Approssimazione di un numero reale con il metodo del minimo scostamento

Il numero reale è approssimato come prodotto di razionali, ogni razionale è un rapporto di due numeri primi successivi scelto in modo da ridurre la differenza tra l'approssimazione attuale e il numero da approssimare:

$$r = \prod_{i=0}^{n} r_i$$

Il numero reale è approssimato come prodotto di razionali in cui il numeratore e il denominatore sono numeri primi scelti secondo la formula:

$$r_0 = 1$$
$$r_i = r_{i-1} \cdot \begin{cases} p_1/p_2 & se\ r_{i-1} > r \\ p_2/p_1 & se\ r_{i-1} < r \end{cases}$$

dove p_1 e p_2, con $p_1 < p_2$, sono due primi successivi.

```
MinimoScostamento(ripetizioni, trascendente)
```

Dove *ripetizioni* è il numero di iterazioni e *trascendente* il numero reale da approssimare. Restituisce un numero reale. Usa *MinimoScostamentoLista(ripetizioni, trascendente)*.

Fattori del prodotto usato nel metodo del minimo scostamento

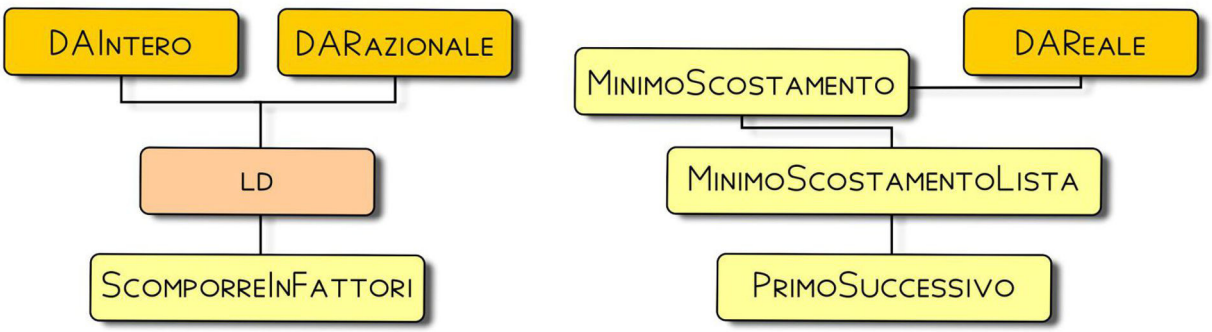
Il metodo del minimo scostamento approssima un numero reale come prodotto di razionali, questa funzione costruisce una lista in cui il primo elemento è *+1* o *−1* a seconda del segno del numero da approssimare e gli elementi successivi sono numeratore e denominatore del razionale seguente.

```
MinimoScostamentoLista(ripetizioni, trascendente)
```

Dove *ripetizioni* è il numero di iterazioni e *trascendente* il numero reale da approssimare. Restituisce una lista di interi.

6.2. Derivata aritmetica

La figura qui di seguito riporta le dipendenze tra le varie funzioni della libreria dedicate al calcolo della derivata aritmetica: le funzioni più in alto nella figura usano le funzioni più in basso a cui sono collegate, come già visto in precedenza.



Derivata logaritmica

La derivata logaritmica, in analogia con la derivata classica, è definita come il rapporto tra la derivata n' di un numero e il numero n stesso:

$$\mathrm{ld}(n) = \frac{n'}{n} = \sum_{p|n} \frac{1}{p}$$

La derivata logaritmica, in pratica, è la somma degli inversi dei primi che dividono n.

```
ld(x)
```

dove x è il numero intero di cui calcolare la derivata logaritmica. Restituisce un numero reale. Usa *ScomporreInFattori* e *PrimoSuccessivo*.

Derivata aritmetica

La derivata di un intero n è data dal numero stesso moltiplicato per la sua derivata logaritmica o, in altre parola, per la somma degli reciproci dei primi che dividono n:

$$n' = n \cdot \mathrm{ld}(n) = n \sum_{p|n} \frac{1}{p}$$

```
DAIntero(x)
```

dove x è il numero intero di cui calcolare la derivata aritmetica. Restituisce un numero intero. Usa *ld*.

La derivata di un numero razionale si ottiene calcolando innanzitutto la somma dei reciproci dei primi che dividono il numeratore meno la somma dei reciproci dei primi che dividono il denominatore e moltiplicando il risultato per il razionale stesso:

$$\left(\frac{n}{d}\right)' = \frac{n}{d} \cdot (\mathrm{ld}(n) - \mathrm{ld}(n)) = \frac{n}{d}\left(\sum_{p|n} \frac{1}{p} - \sum_{p|d} \frac{1}{p}\right)$$

```
DARazionale(numeratore, denominatore)
```

dove *numeratore* è il numeratore del numero razionale da derivare e *denominatore* il denominatore dello stesso numero. Restituisce un numero reale. Usa ld

La derivata di un numero reale è ottenuta sviluppando il numero in un prodotto di frazioni, in cui numeratore e denominatore sono numeri primi, con il metodo del minimo scostamento e successivamente derivando il prodotto così ottenuto. Ovviamente il risultato dipende dall'estensione dello sviluppo.

$$x' = x \cdot \left(\sum_{p|n} \frac{1}{p} - \sum_{p|d} \frac{1}{p}\right)$$

dove la prima somma è estesa ai primi che sono al numeratore dello sviluppo del numero reale e la seconda somma è estesa ai primi che sono al denominatore.

```
DAReale(x, iterazioni)
```

dove x è il numero reale da derivare e *iterazioni* il numero di iterazioni nello sviluppo del numero reale con il metodo del minimo scostamento. Restituisce un numero reale. Usa *DAIntero* e *MinimoScostamentoLista*.

LICEO SCIENTIFICO GRASSI LATINA

CNR IAC
Istituto per le Applicazioni del Calcolo

CNR IFN
Istituto di Fotonica e Nanotecnologie

INM

ISMAR
Istituto di Scienze Marine

LSS G.B. Grassi

LICEO SCIENTIFICO STATALE G.B. GRASSI DI LATINA

WWW.LICEOGRASSILATINA.ORG

CNR - IAC

ISTITUTO PER LE APPLICAZIONI DEL CALCOLO MAURO PICONE

WWW.IAC.CNR.ORG

CNR - IFN Roma

ISTITUTO DI FOTONICA E NANOTECNOLOGIE

WWW.ROMA.IFN.CNR.ORG

CNR - INM

ISTITUTO DI INGEGNERIA DEL MARE

WWW.INSEAN.CNR.ORG

CNR - ISMAR

ISTITUTO DI SCIENZE MARINE

WWW.ISMAR.CNR.ORG