Iniziamo abilitando subito l'estensione `autoreload`

In [1]:
```
%load_ext autoreload
%autoreload 2
```

Equazioni Differenziali Ordinarie

Old News

Qualche anno fa, questi signori hanno vinto un premio Nobel

Sono Michael Rosbash, Jeffrey C. Hall, and Michael W. Young

Ritmo Circadiano

Ma cosa hanno scoperto?

Hanno identificato i meccanismi fondamentali dietro il ritmo circadiano

- Il ritmo circadiano è il nostro "orologio biologico"
- Ha un periodo di circa 24 ore (circa-diem)
- Può adattarsi a segnali esterni (e.g. luce)

Più in dettaglio, come funziona?

- Una proteina "X" si accumula durante la notte...
- ...E viene smaltita durante il giorno

Si tratta di una reazione è autocatalitica!

- La proteina "X" è contemporaneamente un reagente ed un prodotto
- ..In particolare, inibisce la propria produzione

Oscillatore di Van der Pol

Il ritmo circadiano è un esempio di oscillatore biologico

- Le vere reazioni che lo caratterizzano sono complesse...
- ...Ma un modello semplificato è alla nostra portata!

L'Oscillatore di Van der Pol è descritto dall'equazione differenziale:

$$\dot{x} = y$$
$$\dot{y} = \mu(1 - x^2)y - x$$

- x è la corrisponde alla proteina, y è ausiliaria
- x e y cambiano nel tempo (i.e. sono funzioni)
- L'equazione ci dice come avviene tale cambiamento
- ...Specificando il gradiente delle due funzioni

Sistemi Dinamici Continui

Prima di andare nel panico, proviamo a vederla così:

$$\dot{x} = y$$
$$\dot{y} = \mu(1 - x^2)y - x$$

- x e y descrivono lo stato del sistema
- Tale stato varia nel tempo

Si tratta di un sistema dinamico!

- Stavolta però lo stato varia in modo continuo nel tempo
- Le derivate ci dicono in che direzione e con quale intensità

Una equazione differenziale descrive un sistema dinamico continuo

Equazioni Differenziali Ordinarie

Si chiama Equazione Differenziale Ordinarie (Ordinary Differential Equation)

...Una equazione nella forma:

$$\dot{x} = f(x, t)$$

- x rappresenta lo stato del sistema, che cambia rispetto alla variabile t
  - I.e. è una funzione (vettoriale) nella forma $x(t)$
  - t rappresenta di solito (ma non sempre) il tempo
- f definisce il gradiente di x
  - Può dipendere sia dallo stato corrente che dal tempo

Si chiama "ordinaria" perché x dipende da una variabile scalare

- Se t è vettoriale, allora si parla di Equazione Differenziale alle Derivate Parziali
- Noi ci limiteremo a trattare le ODE

Problema ai Valori Iniziali

Quando si risolve una ODE, l'obiettivo è determinare x

Ma x è una funzione! Quindi vogliamo determinarne l'andamento

- Per farlo, è necessario fare alcune assunzioni addizionali...
- ...Per esempio, possiamo specificare un valore iniziale

In questo modo ottieniamo un problema ai valori inziali:

$$\dot{x} = f(t, x)$$
$$x(0) = x_0$$

Intuitivamente, si tratta di simulare il sistema dinamico

- Abbiamo già visto qualcosa di simile!
- ...Solo che finora abbiamo discusso solo sistemi dinamici discreti

Metodo di Eulero

Come possiamo risolvere una ODE?

$$\dot{x} = f(x, t)$$

- Negli esami di Analisi Matematica si studiano approcci simbolici
- Noi vedremo invece come ottenere un metodo numerici

Per prima cosa, dividiamo t in passi discreti di lunghezza h

...Quindi rimpiazziamo $\dot{x}$ con l'approssimazione discreta che abbiamo discusso

$$\frac{x(t+h) - x(t)}{h} = f(x(t), t)$$

- Se h è abbastanza piccolo, dovremmo ottenere una buona approssimazione

Metodo di Eulero

Poiché abbiamo diviso t in una sequenza discreta $\{t_k\}_{k=0}^{n}$

...Ci interessano i valori di x solo in corrispondenza di tali passi

- Quindi possiamo rimpiazzare $x(t)$ con una sequenza di stati $\{x_k\}_{k=0}^{n}$
- Se $x(t)$ corrisponde a x_k, allora $x(t+h)$ corrisponde a x_{k+1}

In questo modo otteniamo

$$\frac{x_{k+1} - x_k}{h} = f(x_k, t_k)$$

E quindi:

$$x_{k+1} = x_k + hf(x_k, t_k)$$

Abbiamo ottenuto un sistema dinamico discreto

Metodo di Eulero

Se i passi discreti non hanno lunghezza uniforme

...La nostra relazione vale ancora sostituendo h con $t_{k+1} - t_k$:

$$x_{k+1} = x_k + (t_{k+1} - t_k)f(x_k, t_k)$$

Possiamo ora risolvere il problema come nel caso discreto

...Ossia con un algoritmo del tipo:

- parametri: $f, x_0, \{t_k\}_{k=0}^n$
- for $k = \{1..n\}$:
  - $x_k = x_{k1} + (t_k - t_{k-1})f(x_{k-1}, t_{k-1})$

Uno dei parametri di ingresso è la funzione f!

Questo algoritmo è noto come metodo di Eulero

Una Possibile Implementazione

Una implementazione del metodo è disponibile in `base.util`

```python
def euler(f, x0, t):
    X = [x0]
    for k in range(1, len(t)):
        nX = X[k-1] + (t[k] - t[k-1]) * f(X[k-1], t[k-1])
        X.append(nX)
    return np.array(X)
```

La nostra implementazione assume che `f` abbia una interfaccia del tipo:

```python
def f(x, t):
    ...
```

- Dove `x` è lo stato (potenzialmente vettoriale) e `t` il tempo corrente
- Per il resto `x0` è lo stato iniziale e `t` è una sequenza di istanti di tempo

Una Applicazione sull'Oscillatore

Proviamo ad applicare il metodo sull'Oscillatore di Van der Pol

L'oscillatore è descritto dall'ODE:

$$\begin{aligned} \dot{x} &= y \\ \dot{y} &= \mu(1 - x^2)y - x \end{aligned}$$

In forma vettoriale possiamo vederla come:

$$\begin{pmatrix} \dot{x} \\ \dot{y} \end{pmatrix} = f((x, y), t) = \begin{pmatrix} y \\ \mu(1 - x^2)y - x \end{pmatrix}$$

- Quindi lo stato è rappresentabile come un vettore con due componenti
- Non c'è in questo caso una dipendenza diretta da t

Una Applicazione sull'Oscillatore

Definiamo la funzione f (nel modulo `base.vdp`)

```
def vdp_dstate(state, t):
    mu = 1
    x, y = state # "spacchetto" lo stato
    dx = y
    dy = mu * (1 - x**2) * y - x
    return np.array([dx, dy])
```

- Assumiamo che lo stato sia passato come un array `state`
- Usiamo l'unpacking per separare x ed y (per chiarezza)
- Calcoliamo le derivate delle due componenti
- Restituiamo un vettore con le due derivate

Una Applicazione sull'Oscillatore

Ora possiamo risolvere l'ODE

In [4]:
```python
import numpy as np
from base import util
from base import vdp

X0 = np.array([1, 1])
t = np.linspace(0, 45, 1000)
X = util.euler(vdp.vdp_dstate, X0, t)
print(X[:4, :])
```

```
[[1.         1.        ]
 [1.04504505 0.95495495]
 [1.08806103 0.90391826]
 [1.12877807 0.84741958]]
```

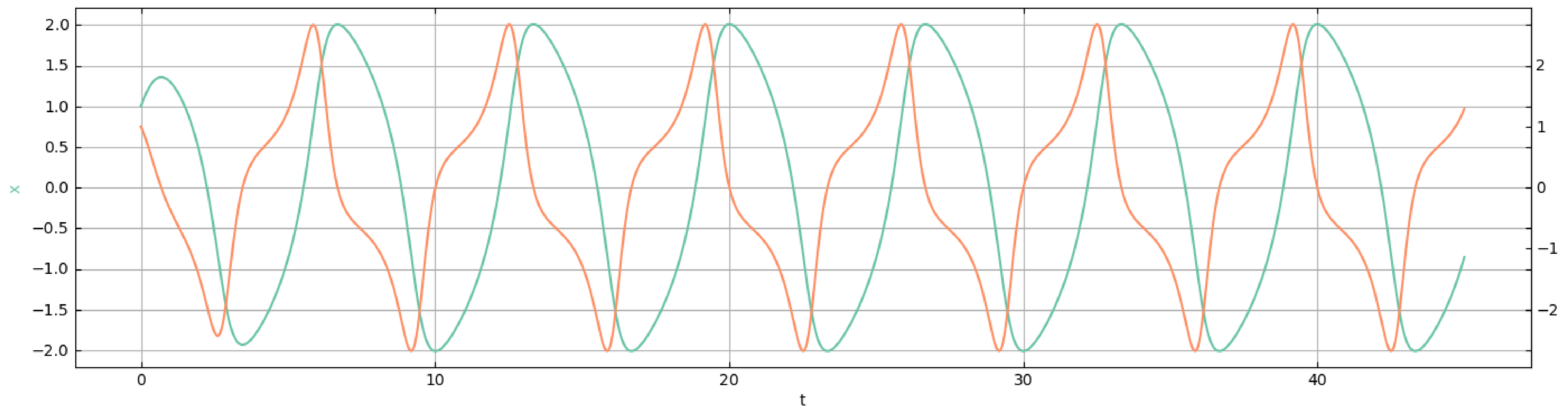
- Come stato iniziale abbiamo $(x, y) = (1, 1)$
- t rappresenta il tempo prende valori (equaspaziati) in $[0, 45]$
- Il risultato `X` è un array con una colonna per ogni componente dello stato
- ...Ed una riga per ogni valutazione effettuata

Una Applicazione sull'Oscillatore

Possiamo disegnare l'evoluzione dello stato nel tempo

Usiamo la funzione `base.util.plot_state_evolution`

In [13]:
```
from base import util
util.plot_state_evolution(X, t, figsize=(20, 4), xlabel='t', ylabels=['x', 'y'])
```



- `x` e `t` sono quelli della nostra funzione `euler`
- La funzione richiede una etichetta `y` per ogni componente dello stato

Utilizzare il Risolutore di `scipy`

Quello di Eulero è il metodo più semplice per risolvere le ODE

...Ma è anche il meno accurato. Ci sono altri metodi!

- E.g. metodo di Eulero inverso, metodi di Runge-Kutta...

Come al solito, non c'è bisogno di re-implementarli

Un buon risolutore di ODE è dato da scipy.integrate.odeint

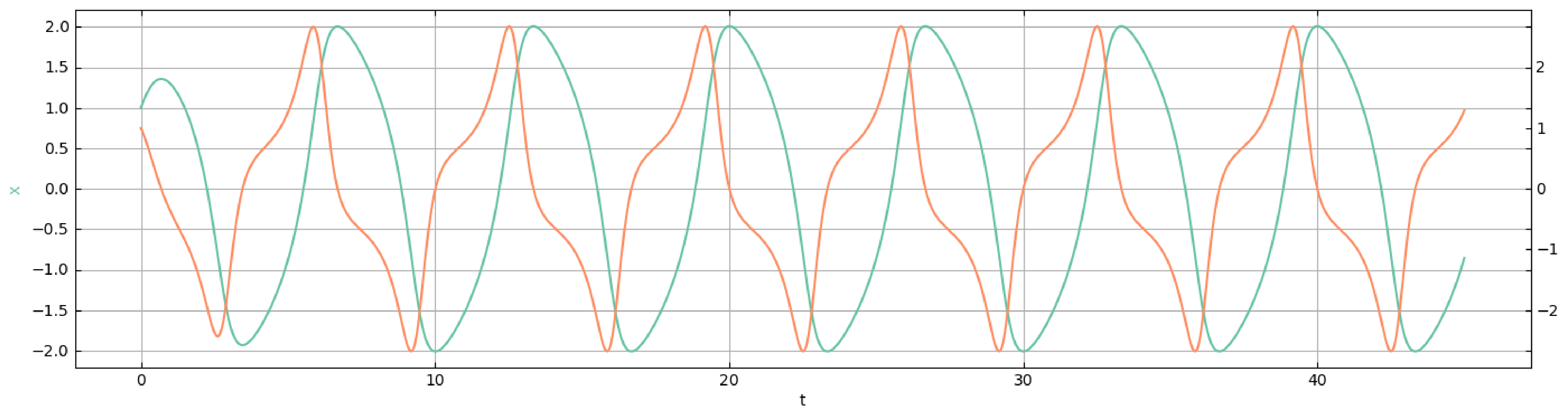
```
def odeint(func, y0, t, ...)
```

- Func è una funzione `f(X, t)` con due parametri (i.e. stato e tempo)
- `y0` è lo stato iniziale
- `t` è il vettore dei tempi da considerare

Utilizzo di `odeint`

Vediamo un esempio di invocazione di `odeint`

In [15]:
```
from scipy.integrate import odeint
X = odeint(vdp.vdp_dstate, X0, t)
util.plot_state_evolution(X, t, figsize=(20, 4), xlabel='t', ylabels=['x', 'y'])
```



- I risultati sono gli stessi di prima (un po' meglio, in realtà!)
- Il comportamento (come previsto) è periodico

Negli esercizi, useremo sempre `odeint`

Valutazione della Funzione Risultato

Diamo un'altra occhiata ai risultati (sia di `euler` che `odeint`)

Abbiamo accesso ai valori di $\{t_k\}_{k=0}^n$

```
In [16]: print(t[:4])
```

```
[0.         0.04504505 0.09009009 0.13513514]
```

...Ed ai valori dello stato corrispondenti

```
In [17]: print(X[:4, :])
```

```
[[1.         1.        ]
 [1.04398574 0.95199307]
 [1.08568168 0.89843058]
 [1.12485152 0.83994828]]
```

Ma se ci interessa il valore dello stato per un t compreso tra t_k e t_{k+1}?

Valutazione della Funzione Risultato

Possiamo usare una semplice interpolazione lineare

...Cioè prendiamo i valori sulla linea tra (x_k, t_k) e (x_{k+1}, t_{k+1})

- È la stessa tecnica che usiamo per disegnare le funzioni!
- ...Ed è disponibile attraverso la funzione numpy.interp

Vediamo un esempio:

La nostra soluzione contiene 1,000 valori per t ed X:

In [18]:
```
print(t.shape, X.shape)
```

```
(1000,) (1000, 2)
```

Recuperiamo 10,000 valori utilizzano l'interpolazione lineare:

In [19]:
```
ts = np.linspace(0, 45, 10000)
xs = np.interp(ts, t, X[:, 0])
```

Valutazione della Funzione Risultato

Cerchiamo di capire meglio l'ultima invocazione

```
In [20]: ts = np.linspace(0, 45, 10000)
         xs = np.interp(ts, t, X[:, 0])
```

- `t` contiene il valore di t per i punti noti
- `X[:, 0]` è il valore della componente x per i punti noti
- `ts` contiene i valori di t per cui vogliamo valutare l'interpolazione

Stampiamo i primi elementi dell'array risultato:

```
In [21]: print(xs[:5])

[1.         1.00439461 1.00878923 1.01318384 1.01757846]
```

NOTA: `interp` richiede che l'array alle ascisse sia ordinato in modo crescente

Parametri Addizionali nelle ODE

Per fornire la possibilità di cambiare i parametri del modello

...Possiamo usare una classe funzione

```
class VdpDstate:
    def __init__(self, mu):
        self.mu = mu

    def __call__(self, state, t):
        mu = self.mu # recupero mu dall'istanza corrente
        x, y = state
        dx = y
        dy = mu * (1 - x**2) * y - x
        return np.array([dx, dy])
```

- Questa soluzione è compatibile con `odeint`
- ...Perché al momento della chiamata vanno passati solo i parametri `x` e `t`

Parametri Addizionali nelle ODE

Vediamo un esempio di utilizzo

In [23]:

```
f = vdp.VdpDstate(mu=0.5)
X = odeint(f, X0, t)
util.plot_state_evolution(X, t, figsize=(20, 4), xlabel='t', ylabels=['x', 'y'])
```