Gaussian_diffusion_W2

August 3, 2024

1 *Diffusion models for Gaussian distributions: Exact solutions and Wasserstein errors* [1]

The following code provides figures and table of the article [1]. You can use it with any covariance matrix (provided eigenvalues can be computed). All details are given to extend our analysis to other numerical schemes.

1.0.1 Reminders of the theory

We consider the Variance preserving (VP) forward process:

$$dx_t = -\beta_t x_t dt + \sqrt{2\beta_t} dw_t, \quad 0 \leq t \leq T, \quad x_0 \sim p_{\text{data}}. \tag{1}$$

Supposing that $p_{\text{data}} = \mathcal{N}(0, \Sigma)$, the law of x_t is $p_t = \mathcal{N}(0, \Sigma_t)$ with

$$\Sigma_t = e^{-2B_t}\Sigma + (1 - e^{-2B_t})I \tag{2}$$

where $B_t = \int_0^t \beta_u$ and consequently the score function verifies $\nabla \log p_t(x) = -\Sigma_t^{-1}x$.

The associated backward SDE is

$$d\tilde{y}_t = \beta_{T-t}(\tilde{y}_t + 2\log p_{T-t}(\tilde{y}_t))dt + \sqrt{2\beta_{T-t}} dw_t, \quad 0 \leq t < T \tag{3}$$

and the reverse flow ODE is

$$d\hat{y}_t = \left[\beta_{T-t}\hat{y}_t + \beta_{T-t}\nabla_{\hat{y}} \log p_{T-t}(\hat{y}_t)\right] dt, \quad 0 \leq t < T. \tag{4}$$

We study the errors of the diffusion models by studying the Wasserstein-2 distance. For two centered Gaussians $\mathcal{N}(0, \Sigma_1)$ and $\mathcal{N}(0, \Sigma_2)$ such that Σ_1, Σ_2 are simultaneously diagonalizable with respective eigenvalues $(\lambda_{i,1})_{1 \leq i \leq d}$ and $(\lambda_{i,2})_{1 \leq i \leq d}$,

$$\mathbf{W}_2(\mathcal{N}(0, \Sigma_2), \mathcal{N}(0, \Sigma_1)) = \sqrt{\sum_{1 \leq i \leq d} (\sqrt{\lambda_{i,1}} - \sqrt{\lambda_{i,2}})^2}. \tag{5}$$

[1]:
```
def W2(lamb1,lamb2) :
    return np.sqrt ( np.sum( (np.sqrt(lamb1)-np.sqrt(lamb2))**2 ) )
```

1.1 Packages

[ ]:
```
!pip install scienceplots
```

[3]:
```
import pylab as plt
import numpy as np
import scienceplots
plt.style.use('science')
from IPython.display import display, Markdown
import os
plt.rcParams.update(plt.rcParamsDefault)
```

2 Load data

The Gaussian distribution is known through the eigenvalues of its covariance matrix Σ. `cifar10.npy` corresponds to the Gaussian distribution fitted to the CIFAR-10 dataset. This the list of empirical covariance eigenvalues of the normalized images of the dataset. `ADSN.npy` corresponds to the ADSN distribution described in the paper.

[24]:
```
PATH_data ='./'

lamb = np.load(PATH_data+'cifar10.npy')
# or lamb = np.load(PATH_data+'ADSN.npy')
```

Bellow, you can choose the outputs path.

[26]:
```
PATH_output = './'
```

2.1 Parametrization

Let consider β_t linear of the form: $\beta_t = t \mapsto \beta_{\min} + (\beta_{\max} - \beta_{\min})\, t$ with $\beta_{\min} = 0.05$ and $\beta_{\max} = 10$. The values are from [2], up to a factor 2 to be consistent with our VP SDE. We introduce also

$$B_t = \int_0^t \beta_u du = \beta_{\min} t + (\beta_{\max} - \beta_{\min}) \frac{t^2}{2}.$$

[27]:
```
T = 1.
beta_min = 0.1/2
beta_max = 20/2

def beta(t) :
    return beta_min + t*(beta_max-beta_min)

def B(t) :
    return beta_min*t +(beta_max-beta_min)*t**2/2
```

3 Forward process

$p_t = \mathcal{N}(0, \Sigma_t)$ with $\Sigma_t = e^{-2B_t}\Sigma + (1 - e^{-2B_t})I$. Let consider the eigenvalues of $(\lambda_i)_{1\leq i\leq d}$ of Σ. Σ_t is diagonalizable in the same orthonormal basis and the ith eigenvalue of Σ_t is

$$\lambda_i^t = e^{-2B_t}\lambda_i + (1 - e^{-2B_t}). \tag{6}$$

[28]:
```
def lamb_Sigma_t(lamb,t) :
    ebt = np.exp(-2*B(t))
    return ebt*lamb+(1-ebt)
```

4 Continuous Initialization error

4.0.1 Continuous SDE

With an initialization $\tilde{y}_0 \sim \mathcal{N}(0, I)$, the solution y_t of Equation (3) follows the law $\tilde{q}_t = \tilde{p}_{T-t}$ where $\tilde{p}_t$ is the Gaussian distribution $\mathcal{N}(0, \tilde{\Sigma}_t)$ and

$$\tilde{\Sigma}_t = \Sigma_t + e^{-2(B_T - B_t)}\Sigma_t^2\Sigma_T^{-1}(\Sigma_T^{-1} - I). \tag{7}$$

Consequently, $\tilde{\Sigma}_t$ is diagonalizable and we can compute its eigenvalues as follows.

[29]:
```
def lamb_SDE_t(lamb,t) :

    lamb_t = lamb_Sigma_t(lamb,t)
    lamb_T = lamb_Sigma_t(lamb,T)

    return lamb_t+np.exp(-2*(B(T)-B(t)))*lamb_t**2/lamb_T*(1/lamb_T-1)
```

4.0.2 Continuous ODE

With an initialization $\hat{y}_0 \sim \mathcal{N}(0, I)$, the solution $\hat{y}_t$ of Equation (4) follows the law $\hat{q}_t = \hat{p}_{T-t}$ where $\hat{p}_t$ is the Gaussian distribution $\mathcal{N}(0, \widehat{\Sigma}_t)$ and

$$\widehat{\Sigma}_t = \Sigma_T^{-1}\Sigma_t. \tag{8}$$

Consequently, $\tilde{\Sigma}_t$ is diagonalizable and we can compute its eigenvalues as follows.

[30]:
```
def lamb_ODE_t(lamb,t) :

    lamb_t = lamb_Sigma_t(lamb,t)
    lamb_T = lamb_Sigma_t(lamb,T)


    return lamb_t/lamb_T
```

5 Discretization of the equations

5.1 Discretization of the bacward SDE

Under Gaussian assumption, Equation (3) becomes:

$$d\tilde{y}_t = \beta_{T-t}(\tilde{y}_t - 2\Sigma_{T-t}^{-1}(\tilde{y}_t))dt + \sqrt{2\beta_{T-t}}dw_t, \quad 0 \leq t < T. \tag{9}$$

We study the Euler-Maruyama's scheme (EM) and the Exponential Integrator scheme (EI).

5.1.1 Euler Maruyama's scheme

The EM discretization of Equation (9) is

$$y^{\mathrm{EM},k+1} = y^{\mathrm{EM},k} + \Delta_t \beta_{T-t_k}\left(y_k - 2\Sigma_{T-t_k}^{-1} y^{\mathrm{EM},k}\right) + \sqrt{2\Delta_t \beta_{T-t_k}} z_k, \quad z_k \sim \mathcal{N}_0. \tag{10}$$

Consequently, the ith eigenvalue $\lambda_i^{EM,k}$ of the covariance matrix of $(y^{k,\mathrm{EM}})_{0\leq k\leq N-1}$ verifies

$$\lambda_i^{\mathrm{EM},k+1} = \left(1 + \Delta_t \beta_{T-t_k}\left(1 - \frac{2}{\lambda_i^{T-t_k}}\right)\right)^2 \lambda_i^{\mathrm{EM},k} + 2\Delta_t \beta_{T-t_k} \tag{11}$$

with λ_i^t ith eigenvalue of Σ_t and $\lambda_i^{\mathrm{EM},0}$ initialized at 1 or λ_i^T depending on the choice of initialization. The following compute the Wasserstein error at each step.

[31]:
```
def W2_EM(N,lamb,t_eps=0,p_T = False,all_t = True) :

    tk = np.array([(T-t_eps)*k/(N-1) for k in range(N)])
    Delta_t = tk[1]-tk[0]

    #Initialization at p_T
    if p_T :
        lamb_EM = lamb_Sigma_t(lamb,T)
    #Initialization at N_0
    else :
        lamb_EM = np.ones_like(lamb)

    if all_t :
        W2_EM_list = [W2(lamb_EM,lamb_Sigma_t(lamb,T))]

    for k in range(N-1) :
        lamb_T_tk = lamb_Sigma_t(lamb,T-tk[k])
        beta_T_tk = beta(T-tk[k])

        lamb_EM =  (1+Delta_t*beta_T_tk*(1-2/lamb_T_tk))**2*lamb_EM +␣
↪2*Delta_t*beta_T_tk

        if all_t :
            W2_EM_list.append(W2(lamb_EM,lamb_Sigma_t(lamb,T-tk[k+1])))
```

```
    if all_t :
        W2_EM_list.reverse()
        return W2_EM_list

    else :
        return W2(lamb_EM,lamb)
```

5.1.2 Exponential Integrator (EI) scheme

The EI discretization of Equation (9) is

$$y^{\mathrm{EI},k+1} = y^{\mathrm{EI},k} + \gamma_{1,k}\left(y^{\mathrm{EI},k+1} - 2\Sigma_{T-t_k}^{-1} y^{\mathrm{EI},k}\right) + \sqrt{2\gamma_{2,k}} z_k \quad z_k \sim \mathcal{N}_0 \tag{12}$$

with $\gamma_{1,k} = \exp\left(B_{T-t_k} - B_{T-t_{k+1}}\right) - 1$ and $\gamma_{2,k} = \frac{1}{2}\left[\exp\left(2\left(B_{T-t_k} - B_{T-t_{k+1}}\right)\right) - 1\right]$.
Consequently, the ith eigenvalue $\lambda_i^{\mathrm{EI},k}$ of the covariance matrix of $\left(y^{k,\mathrm{EI}}\right)_{0\leq k\leq N-1}$ verifies

$$\lambda_i^{\mathrm{EI},k+1} = \left(1 + \gamma_{1,k}\left(1 - \frac{2}{\lambda_i^{T-t_k}}\right)\right)^2 \lambda_i^{\mathrm{EI},k} + 2\gamma_{2,k} \tag{13}$$

with λ_i^t ith eigenvalue of Σ_t and $\lambda_i^{\mathrm{EI},0}$ initialized at 1 or λ_i^T depending on the choice of initialization. The following compute the Wasserstein error at each step.

[32]:
```
def W2_EI(N,lamb,t_eps=0,p_T = False,all_t = True) :

    tk = np.array([(T-t_eps)*k/(N-1) for k in range(N)])
    Delta_t = tk[1]-tk[0]

    #Initialization at p_T
    if p_T :
        lamb_EI = lamb_Sigma_t(lamb,T)
    #Initialization at N_0
    else :
        lamb_EI = np.ones_like(lamb)

    if all_t :
        W2_EI_list = [W2(lamb_EI,lamb_Sigma_t(lamb,T))]

    for k in range(N-1) :
        gamma_1_k = np.exp(B(T-tk[k])-B(T-tk[k+1]))-1
        gamma_2_k = (np.exp(2*(B(T-tk[k])-B(T-tk[k+1])))-1)/2

        lamb_T_tk =  lamb_Sigma_t(lamb,T-tk[k])

        beta_T_tk = beta(T-tk[k])
```

```
        lamb_EI = (1+gamma_1_k*(1-2/lamb_T_tk))**2*lamb_EI + 2*gamma_2_k

        if all_t :

            W2_EI_list.append(W2(lamb_EI,lamb_Sigma_t(lamb,T-tk[k+1])))

    if all_t :
        W2_EI_list.reverse()

        return W2_EI_list

    else :

        return W2(lamb_EI,lamb)
```

5.2 Discretization of the flow ODE

Under Gaussian assumption, Equation (4) becomes

$$d\hat{y}_t = \left[\beta_{T-t}\hat{y}_t - \beta_{T-t}\Sigma_{T-t}^{-1}(\hat{y}_t)\right] dt, \quad 0 \le t < T. \tag{14}$$

We study the Euler scheme and the Heun's scheme.

5.3 Euler scheme

The EM discretization of Equation (14) is

$$y^{\mathrm{Euler},k+1} = y^{\mathrm{Euler},k} + \Delta_t \beta_{T-t_k}\left(y^{\mathrm{Euler},k} - \Sigma_{T-t_k}^{-1} y^{\mathrm{Euler},k}\right). \tag{15}$$

Consequently, the ith eigenvalue $\lambda_i^{Euler,k}$ of the covariance matrix of $\left(y^{k,\mathrm{Euler}}\right)_{0\le k\le N-1}$ verifies

$$\lambda_i^{\mathrm{Euler},k+1} = \left(1 + \Delta_t \beta_{T-t_k}\left(1 - \frac{1}{\lambda_i^{T-t_k}}\right)\right)^2 \lambda_i^{\mathrm{Euler},k} \tag{16}$$

with λ_i^t ith eigenvalue of Σ_t and $\lambda_i^{\mathrm{Euler},0}$ initialized at 1 or λ_i^T depending on the choice of initialization. The following compute the Wasserstein error at each step.

[33]:
```
def W2_Euler(N,lamb,t_eps=0,p_T = False,all_t = True) :

    tk = np.array([(T-t_eps)*k/(N-1) for k in range(N)])
    Delta_t = tk[1]-tk[0]

    #Initialization at p_T
    if p_T :
        lamb_Euler = lamb_Sigma_t(lamb,T)
    #Initialization at N_0
```

```
    else :
        lamb_Euler = np.ones_like(lamb)

    if all_t :
        W2_Euler_list = [W2(lamb_Euler,lamb_Sigma_t(lamb,T))]

    for k in range(N-1) :
        lamb_T_tk =  lamb_Sigma_t(lamb,T-tk[k])
        beta_T_tk = beta(T-tk[k])

        lamb_Euler = (1+Delta_t*beta_T_tk*(1-1/lamb_T_tk))**2*lamb_Euler

        if all_t :
            W2_Euler_list.append(W2(lamb_Euler,lamb_Sigma_t(lamb,T-tk[k+1])))

    if all_t :
        W2_Euler_list.reverse()

        return W2_Euler_list

    else :

        return W2(lamb_Euler,lamb)
```

5.4 Heun's scheme

The EM discretization of Equation (14) is

$$y^{k+1/2,\mathrm{Heun}} = y^{k,\mathrm{Heun}} + \Delta_t \beta_{T-t_k} \left(y^{k,\mathrm{Heun}} - \Sigma_{T-t_k}^{-1} y^{k,\mathrm{Heun}} \right)$$
$$y^{k+1,\mathrm{Heun}} = y^{k,\mathrm{Heun}} + \frac{\Delta_t}{2} \beta_{T-t_k} \left(y^{k,\mathrm{Heun}} - \Sigma_{T-t_k}^{-1} y^{k,\mathrm{Heun}} \right) + \frac{\Delta_t}{2} \beta_{T-t_{k+1}} \left(y^{k+1/2,\mathrm{Heun}} - \Sigma_{T-t_{k+1}}^{-1} y^{k+1/2,\mathrm{Heun}} \right).$$

Consequently, the ith eigenvalue $\lambda_i^{\mathrm{Heun},k}$ of the covariance matrix of $(y^{k,\mathrm{Heun}})_{0 \le k \le N-1}$ verifies

$$\lambda_i^{k+1,\mathrm{Heun}} = \left(1 + \frac{\Delta_t}{2} \beta_{T-t_k} \left(1 - \frac{1}{\lambda_i^{T-t_k}} \right) + \frac{\Delta_t}{2} \beta_{T-t_{k+1}} \left(1 - \frac{1}{\lambda_i^{T-t_{k+1}}} \right) \left(1 + \Delta_t \beta_{T-t_k} \left(1 - \frac{1}{\lambda_i^{T-t_k}} \right) \right) \right)^2 \lambda_i^{k,\mathrm{Heun}}$$

with λ_i^t ith eigenvalue of Σ_t. With $\lambda_i^{\mathrm{Heun},0}$ initialized at 1 or λ_i^T depending on the choice of initialization.

[34]:
```
def W2_Heun(N,lamb,t_eps=0,p_T = False,all_t = True) :
    tk = np.array([(T-t_eps)*k/(N-1) for k in range(N)])
    Delta_t = tk[1]-tk[0]

    #Initialization at p_T
    if p_T :
```

```
        lamb_Heun = lamb_Sigma_t(lamb,T)
    #Initialization at N_0
    else :
        lamb_Heun = np.ones_like(lamb)

    if all_t :
        W2_Heun_list = [W2(lamb_Heun,lamb_Sigma_t(lamb,T))]

    for k in range(N-1) :
        lamb_T_tk =  lamb_Sigma_t(lamb,T-tk[k])
        beta_T_tk = beta(T-tk[k])

        lamb_T_tk_1 =  lamb_Sigma_t(lamb,T-tk[k+1])
        beta_T_tk_1 = beta(T-tk[k+1])


        lamb_Heun =(1+ Delta_t/2*beta_T_tk*(1-1/lamb_T_tk)+Delta_t/
↪2*beta_T_tk_1*(1-1/lamb_T_tk_1)*(1+Delta_t*beta_T_tk*(1-1/
↪lamb_T_tk)))**2*lamb_Heun

        if all_t :
            W2_Heun_list.append(W2(lamb_Heun,lamb_T_tk_1 ))

    if all_t :
        W2_Heun_list.reverse()

        return W2_Heun_list

    else :

        return W2(lamb_Heun,lamb)
```

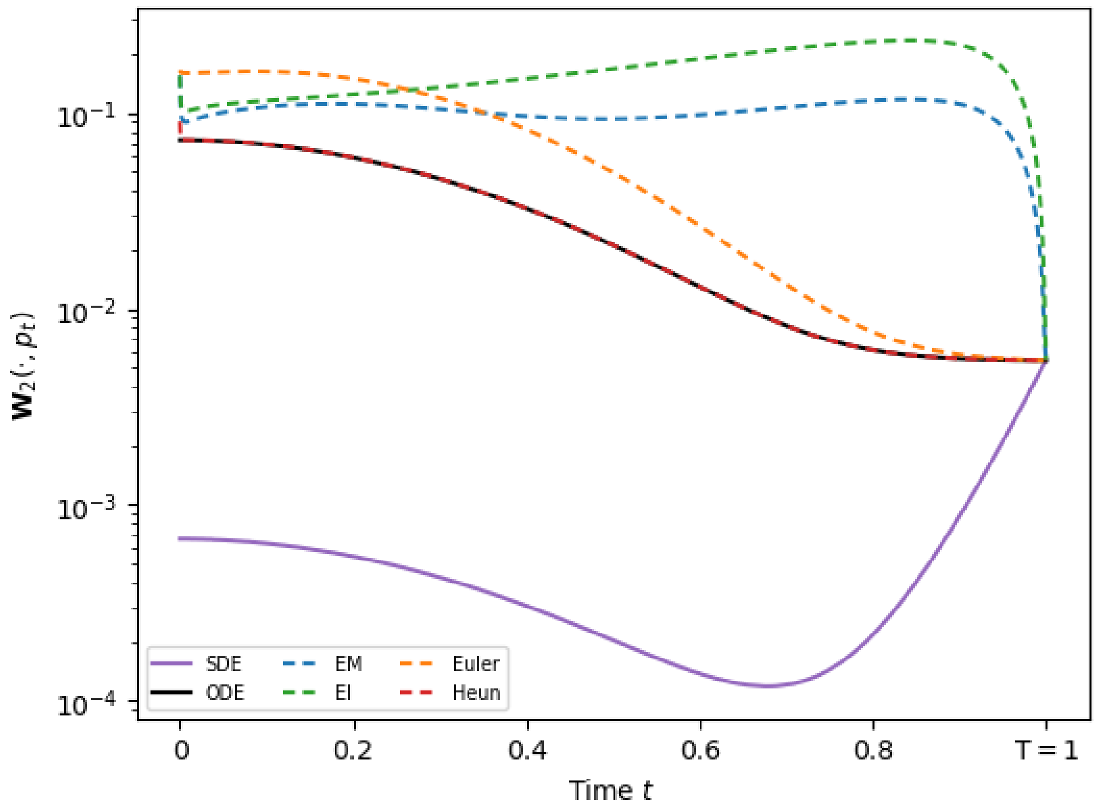
6 Error graphs

Bellow, the code to plot the Figure 1 of [1] showing the Wasserstein value of the discretization, the initialization along the time and the truncation error.

6.0.1 Discretization and initialization

[35]:
```
T = 1
N = 1000
tk = np.array([T*k/(N-1) for k in range(N)])


W2_SDE = [W2(lamb_SDE_t(lamb,T-t),lamb_Sigma_t(lamb,T-t)) for t in tk]
W2_SDE.reverse()
```

```
W2_ODE = [W2(lamb_ODE_t(lamb,T-t),lamb_Sigma_t(lamb,T-t)) for t in tk]
W2_ODE.reverse()

plt.semilogy(tk,W2_SDE,'-',label='SDE',color='tab:purple')
plt.semilogy(tk,W2_ODE,'-',label='ODE',color='k')

das = (3,2)
plt.semilogy(tk,W2_EM(N,lamb),'--',label='EM',dashes=das,color='C0')
plt.semilogy(tk,W2_EI(N,lamb),'--',label='EI',dashes=das,color='tab:green')
plt.semilogy(tk,W2_Euler(N,lamb),'--',label='Euler',dashes=das,color='tab:
 ↪orange')
plt.semilogy(tk,W2_Heun(N,lamb),'--',label='Heun',dashes=das,color='tab:red')



plt.ylabel('$\mathbf{W}_2(\cdot,p_t)$')
axes = plt.gca()
axes.xaxis.set_ticks([0,0.2,0.4,0.6,0.8,1.0])
axes.xaxis.set_ticklabels(["0","0.2","0.4","0.6","0.8",r"$\mathrm{T} = 1$"])

plt.xlabel('Time $t$')


plt.legend( ncol=3,fontsize='x-small')

plt.show()

plt.savefig(PATH_output+'discretization_initialization_error.pdf',␣
 ↪bbox_inches='tight', dpi=100)
```




```
<Figure size 640x480 with 0 Axes>
```

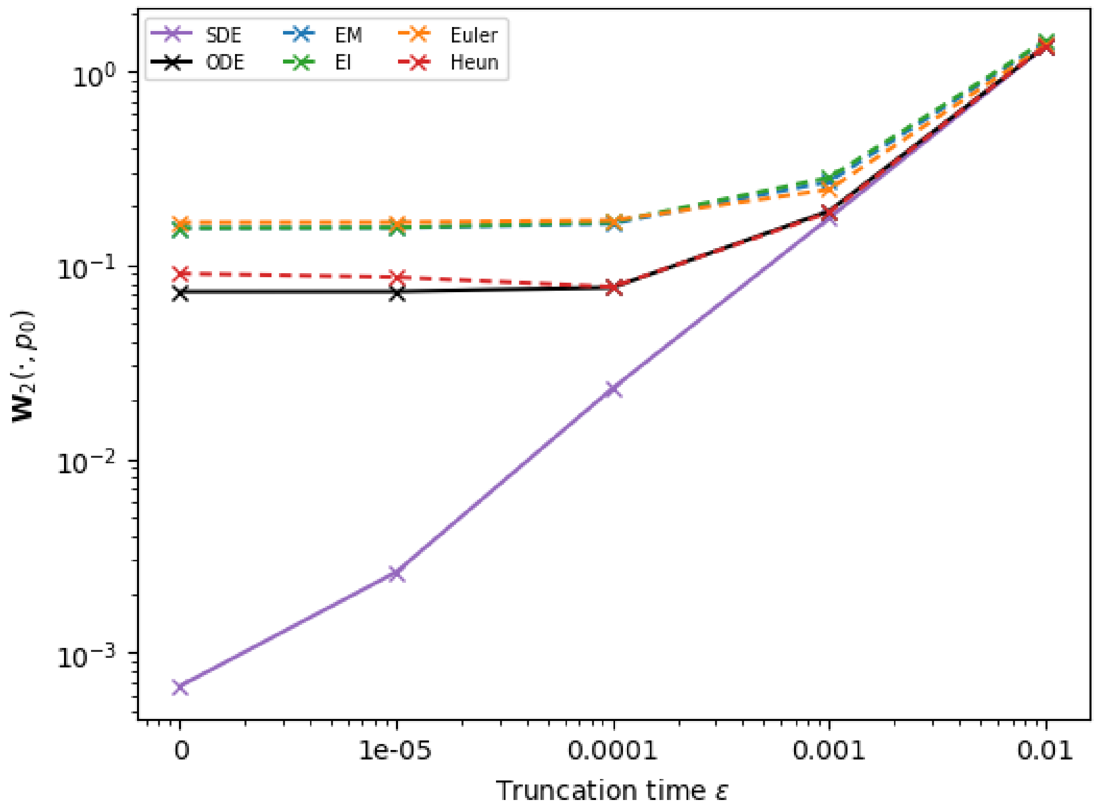
6.0.2 Truncation

[36]:
```
eps_list_graph = [0,10**-5,10**-4,10**-3,10**-2]

W2_eps_EM = []
W2_eps_EI = []
W2_eps_Euler = []
W2_eps_Heun = []
W2_eps_SDE = []
W2_eps_ODE = []

N = 1000

eps_list_graph = [0,10**-5,10**-4,10**-3,10**-2]
for t_eps in eps_list_graph :
    W2_eps_EM.append(W2_EM(N,lamb,t_eps,all_t=False))
    W2_eps_EI.append(W2_EI(N,lamb,t_eps,all_t=False))
    W2_eps_Euler.append(W2_Euler(N,lamb,t_eps,all_t=False))
    W2_eps_Heun.append(W2_Heun(N,lamb,t_eps,all_t=False))
```

```
    W2_eps_SDE.append(W2(lamb_SDE_t(lamb,t_eps),lamb_Sigma_t(lamb,0)))
    W2_eps_ODE.append(W2(lamb_ODE_t(lamb,t_eps),lamb_Sigma_t(lamb,0)))
```

[37]:

```
plt.figure()

T_eps_plot = [10**-6,10**-5,10**-4,10**-3,10**-2]
#The 'zero eps' is plotted at 10**-6 to enable the loglog setting

plt.loglog(T_eps_plot,W2_eps_SDE,'-x',label='SDE',color='tab:purple')
plt.loglog(T_eps_plot,W2_eps_ODE,'-x',label='ODE',color='k')
das = (3,2)
plt.loglog(T_eps_plot,W2_eps_EM,'--x',label='EM',dashes=das,color='C0')
plt.loglog(T_eps_plot,W2_eps_EI,'--x',label='EI',dashes=das,color='tab:green')
plt.loglog(T_eps_plot,W2_eps_Euler,'--x',label='Euler',dashes=das,color='tab:
 ↪orange')
plt.loglog(T_eps_plot,W2_eps_Heun,'--x',label='Heun',dashes=das,color='tab:red')

#plt.legend()
axes = plt.gca()
axes.xaxis.set_ticks(T_eps_plot)
axes.xaxis.set_ticklabels([str(t_eps) for t_eps in eps_list_graph])
plt.ylabel('$\mathbf{W}_2(\cdot,p_0)$')

plt.xlabel(r'Truncation time $\varepsilon$')



plt.legend(ncol=3,fontsize='x-small')

plt.show()

plt.savefig(PATH_output+'truncation_error.pdf', bbox_inches='tight', dpi=100)
```

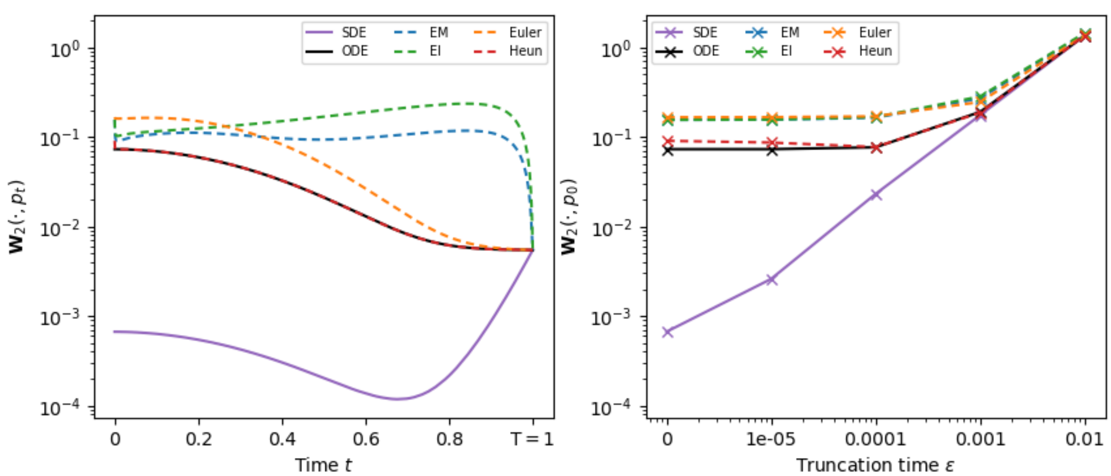



```
<Figure size 640x480 with 0 Axes>
```

6.1 Two graphs with the same scale

[38]:
```
plt.figure(figsize=(10,4))
plt.subplot(1,2,1)
plt.semilogy(tk,W2_SDE,'-',label='SDE',color='tab:purple')
plt.semilogy(tk,W2_ODE,'-',label='ODE',color='k')

das = (3,2)
plt.semilogy(tk,W2_EM(N,lamb),'--',label='EM',dashes=das,color='C0')
plt.semilogy(tk,W2_EI(N,lamb),'--',label='EI',dashes=das,color='tab:green')
plt.semilogy(tk,W2_Euler(N,lamb),'--',label='Euler',dashes=das,color='tab:
↪orange')
plt.semilogy(tk,W2_Heun(N,lamb),'--',label='Heun',dashes=das,color='tab:red')


plt.ylabel('$\mathbf{W}_2(\cdot,p_t)$')
axes = plt.gca()
axes.xaxis.set_ticks([0,0.2,0.4,0.6,0.8,1.0])
axes.xaxis.set_ticklabels(["0","0.2","0.4","0.6","0.8",r"$\mathrm{T} = 1$"])
```

```
plt.xlabel('Time $t$')


plt.legend( ncol=3,fontsize='x-small')

plt.subplot(1,2,2,sharey=axes)

T_eps_plot = [10**-6,10**-5,10**-4,10**-3,10**-2]


plt.loglog(T_eps_plot,W2_eps_SDE,'-x',label='SDE',color='tab:purple')
plt.loglog(T_eps_plot,W2_eps_ODE,'-x',label='ODE',color='k')
das = (3,2)
plt.loglog(T_eps_plot,W2_eps_EM,'--x',label='EM',dashes=das,color='C0')
plt.loglog(T_eps_plot,W2_eps_EI,'--x',label='EI',dashes=das,color='tab:green')
plt.loglog(T_eps_plot,W2_eps_Euler,'--x',label='Euler',dashes=das,color='tab:
 ↪orange')
plt.loglog(T_eps_plot,W2_eps_Heun,'--x',label='Heun',dashes=das,color='tab:red')

axes = plt.gca()
axes.xaxis.set_ticks(T_eps_plot)
axes.xaxis.set_ticklabels([str(t_eps) for t_eps in eps_list_graph])
plt.ylabel('$\mathbf{W}_2(\cdot,p_0)$')

plt.xlabel(r'Truncation time $\varepsilon$')

plt.legend(ncol=3,fontsize='x-small')

plt.show()

plt.savefig(PATH_output+'truncation_discretization_initialization.pdf',␣
 ↪bbox_inches='tight', dpi=100)
```



```
<Figure size 640x480 with 0 Axes>
```

7 Ablation study table

The following code displays the table corresponding to Table 2 of [1].

[39]:
```
N_list = [50,250,500,1000] #500
eps_list = [0., 10**-5,10**-3,10**-2]
P_T = [True,False]
schemes_list = ['EM','EI','Euler','Heun']


W2_dict = {scheme: {str(p_T):{} for p_T in P_T } for scheme in schemes_list}
W2_dict['SDE'] =  {str(p_T):{} for p_T in P_T }
W2_dict['ODE'] = {str(p_T):{} for p_T in P_T }
#Continuous integration


for scheme in ['SDE','ODE'] :
    if scheme == 'SDE' :
        lamb_funct = lamb_SDE_t
    elif scheme == 'ODE' :
        lamb_funct = lamb_ODE_t
    for t_eps in eps_list :
        W2_dict[scheme]['False'][str(t_eps)] =␣
 ↪W2(lamb_funct(lamb,t_eps),lamb_Sigma_t(lamb,0))
        W2_dict[scheme]['True'][str(t_eps)] =␣
 ↪W2(lamb_Sigma_t(lamb,t_eps),lamb_Sigma_t(lamb,0))


for scheme in schemes_list :

    if scheme == 'EM' :
            W2_funct = W2_EM
    elif scheme == 'EI' :
            W2_funct = W2_EI
    elif scheme == 'Euler' :
            W2_funct = W2_Euler
    elif scheme == 'Heun' :
            W2_funct = W2_Heun
    for p_T in P_T :
        for N in N_list :
            W2_dict[scheme][str(p_T)][str(N)] = {}
            for t_eps in eps_list :
```

```
                W2_dict[scheme][str(p_T)][str(N)][str(t_eps)] =␣
↪W2_funct(N,lamb,t_eps=t_eps,p_T = p_T ,all_t=False)
```

7.0.1 Markdown table

In the following, the table is displayed in the notebook via Markdown.

[40]:
```
def formatting_number(x) :
    if x == 0 :
        str_x = '0'
    elif x == np.inf :
        str_x = '-'
    elif np.log10(x).is_integer() :
        str_x = '10^{'+str(int(np.log10(x)))+'}'
    elif  x > 10**2 :
        str_x = '{:1.1E}'.format(x)
    elif x < 10**-2 :
        str_x =  '{:1.1E}'.format(x)
    else :
        str_x =  '{:1.2f}'.format(x)
    return str_x
```

[41]:
```
table_Markdown = '|||'
table_Markdown += ' Continuous||'

for N in N_list :
    table_Markdown += '  N =  ' +str(N)+'||'

table_Markdown += '\n'

table_Markdown += '|:---:|:---:|:---:|:---:|'
for N in N_list :
    table_Markdown += ':---:|:---:|'

table_Markdown += '\n'

table_Markdown += '|||$p_T$|$\mathcal{N}_0$|'
for N in N_list :
    table_Markdown += '$p_T$|$\mathcal{N}_0$|'

table_Markdown += '\n'


for scheme in schemes_list :
    table_Markdown += '|'+scheme+'|'
    for t_eps in eps_list :
        if t_eps != eps_list[0] :
```

```
            table_Markdown += '||'
        table_Markdown += r'$\varepsilon = '+formatting_number(t_eps)+'$|'
        if scheme in ['EM','EI'] :
            table_Markdown += formatting_number(W2_dict['SDE']['True'][str(t_eps)])+'|'
            table_Markdown += formatting_number(W2_dict['SDE']['False'][str(t_eps)])+'|'
        if scheme in ['Euler','Heun'] :
            table_Markdown += formatting_number(W2_dict['ODE']['True'][str(t_eps)])+'|'
            table_Markdown += formatting_number(W2_dict['ODE']['False'][str(t_eps)])+'|'
        for N in N_list :
          table_Markdown += formatting_number(W2_dict[scheme]['True'][str(N)][str(t_eps)])+'|'
          table_Markdown += formatting_number(W2_dict[scheme]['False'][str(N)][str(t_eps)])+'|'
        table_Markdown += '\n'



Markdown(table_Markdown)
```

[41]:

		Continuous		N = 50		N = 250		N = 500		N = 1000	
		p_T	$\mathcal{N}_0$	p_T	$\mathcal{N}_0$	p_T	$\mathcal{N}_0$	p_T	$\mathcal{N}_0$	p_T	$\mathcal{N}_0$
EM	$\varepsilon = 0$	0	6.7E-04	4.77	4.77	0.65	0.65	0.31	0.31	0.15	0.16
	$\varepsilon = 10^{-5}$	2.5E-03	2.6E-03	4.77	4.77	0.65	0.65	0.31	0.31	0.16	0.16
	$\varepsilon = 10^{-3}$	0.17	0.17	4.67	4.67	0.69	0.69	0.39	0.39	0.27	0.27
	$\varepsilon = 10^{-2}$	1.35	1.35	4.56	4.56	1.69	1.69	1.50	1.50	1.42	1.42
EI	$\varepsilon = 0$	0	6.7E-04	2.81	2.81	0.57	0.57	0.30	0.30	0.16	0.16
	$\varepsilon = 10^{-5}$	2.5E-03	2.6E-03	2.81	2.81	0.57	0.57	0.30	0.30	0.16	0.16
	$\varepsilon = 10^{-3}$	0.17	0.17	2.91	2.91	0.66	0.66	0.41	0.41	0.28	0.28
	$\varepsilon = 10^{-2}$	1.35	1.35	3.93	3.93	1.76	1.76	1.55	1.55	1.45	1.45
Euler	$\varepsilon = 0$	0	0.07	1.72	1.78	0.38	0.44	0.19	0.26	0.10	0.17
	$\varepsilon = 10^{-5}$	2.5E-03	0.07	1.72	1.78	0.38	0.44	0.20	0.26	0.10	0.17

		Continuous		N = 50		N = 250		N = 500		N = 1000	
	$\varepsilon = 10^{-3}$	0.17	0.19	1.72	1.78	0.42	0.48	0.27	0.32	0.21	0.25
	$\varepsilon = 10^{-2}$	1.35	1.36	2.21	2.25	1.41	1.43	1.37	1.38	1.36	1.37
Heun	$\varepsilon = 0$	0	0.07	7.09	7.09	0.72	0.73	0.21	0.22	0.05	0.09
	$\varepsilon = 10^{-5}$	2.5E-03	0.07	6.48	6.48	0.64	0.65	0.18	0.20	0.05	0.09
	$\varepsilon = 10^{-3}$	0.17	0.19	0.56	0.57	0.13	0.15	0.16	0.18	0.17	0.19
	$\varepsilon = 10^{-2}$	1.35	1.36	1.37	1.38	1.35	1.36	1.35	1.36	1.35	1.36

7.0.2 Tex table

In the following, a table.tex is created and compiled (if pdflatex is available) to obtain Table 2 of [1].

[42]:
```python
output_tex = PATH_output + 'table.tex'

#preamble
table_tex  = r'\documentclass{article}' + '\n'
table_tex  += '\n'
table_tex  += r'\usepackage{booktabs}' + '\n'
table_tex  += r'\usepackage{multirow}' + '\n'
table_tex  += r'\usepackage{graphicx}' + '\n'
table_tex  +='\n'
table_tex  += r'\begin{document}' + '\n'
table_tex  +='\n'
#Table
table_tex  += r'\begin{table}' + '\n'
table_tex += r'\centering' + '\n'
table_tex += r'\begin{tabular}{'
table_tex += 'l'*(len(N_list)*len(P_T)+4) + '} \n'
table_tex += r'\toprule' + '\n'

table_tex += r'&'
table_tex += r' &\multicolumn{2}{c}{Continuous}' + '\n'

for N in N_list :
    table_tex += r'& \multicolumn{2}{c}{$N = '+str(N)+ r'$}' + '\n'

table_tex += r'\\' +'\n'

for k in range(2,len(N_list)*len(P_T)+4,2) :
    table_tex += r'\cmidrule(lr){'+str(k+1) +'-'+str(k+2)+ r'}' + '\n'
```

```python
table_tex += r'& &$p_T$ & $\mathcal{N}_0$ & ' + '\n'
for N in N_list :
    if N == N_list[-1] :
        table_tex += r'$p_T$ & $\mathcal{N}_0$ \\' + '\n'
    else :
        table_tex += r'$p_T$ & $\mathcal{N}_0$ &' + '\n'

table_tex += r'\midrule' + '\n'

for scheme in schemes_list :
    table_tex += r'\parbox[t]{2mm}{\multirow{4}{*}{\rotatebox[origin=c]{90}{'+scheme+r'}}}' + '\n'
    #Continuous column
    for t_eps in eps_list :
        table_tex += r'& \multicolumn{1}{|l}{$\varepsilon = '+formatting_number(t_eps)+'$}\n'

        if scheme in ['EM','EI'] :
            table_tex += ' & '+formatting_number(W2_dict['SDE']['True'][str(t_eps)])+'\n'
            table_tex += ' & '+formatting_number(W2_dict['SDE']['False'][str(t_eps)])+'\n'
        if scheme in ['Euler','Heun'] :
            table_tex += ' & '+formatting_number(W2_dict['ODE']['True'][str(t_eps)])+'\n'
            table_tex += ' & '+formatting_number(W2_dict['ODE']['False'][str(t_eps)])+'\n'
        for N in N_list :
            table_tex += ' & '+formatting_number(W2_dict[scheme]['True'][str(N)][str(t_eps)])+'\n'
            table_tex += ' & '+formatting_number(W2_dict[scheme]['False'][str(N)][str(t_eps)])+'\n'
        table_tex +=  r'\\' + '\n'

table_tex += r'\bottomrule' + '\n'
table_tex += r'\end{tabular}' + '\n'
table_tex += r'\end{table}' + '\n'

table_tex += r'\end{document}'

if os.path.exists(output_tex) :
    os.remove(output_tex)

f = open(output_tex, "a")
```

```
f.write(table_tex)
f.close()
```

The following cell compiles the file .tex if latex is available.

[ ]:
```
from distutils.spawn import find_executable
if find_executable('latex'):
    os.system('pdflatex -v '+output_tex)
```

8 Bibliography

[1] (2024). Diffusion models for Gaussian distributions: Exact solutions and Wasserstein errors. Preprint.

[2] Yang Song, Jascha Sohl-Dickstein, Diederik P. Kingma, Abhishek Kumar, Stefano Ermon, Ben Poole (2021). Score-Based Generative Modeling through Stochastic Differential Equations. ICLR

[ ]:
```

```