# assignment1_sol

January 28, 2025

## 1 Assignment 1 Solutions

Consider the harmonic oscillator with Lagrangian,

$$L(x,\dot{x}) = K(\dot{x}) - V(x) = \frac{1}{2}\dot{x}^2 - \frac{1}{2}x^2,$$

where $x$ is the position of the oscillator and $\dot{x}$ is its velocity. Note this is expressed in units where the mass $m = 1$ and angular frequency $\omega = 1$, so the classical oscillator period $T_0 = 2\pi$. For this problem, you can work in units $m = \omega = \hbar = 1$, so the classical oscillator period $T_0 = 2\pi$.

We will use the discrete approximation to the path integral for the harmonic oscillator, where the time step is $\epsilon = \Delta t = T_0/128$. The electron position is also discretized into $N_D + 1$ possible points, $x_0 = -4, x_1, x_2, \ldots, x_{N_D} = +4$, where $N_D = 600$. The initial probability amplitude (sometimes called the wavefunction) of the electron is a Gaussian centered at $x_{\text{start}}$,

$$\Psi_0(x) = \left(\frac{\alpha}{\pi}\right)^{1/4} \exp\left(-\frac{\alpha}{2}(x - x_{\text{start}})^2\right),$$

where $\alpha = 2$ and $x_{\text{start}} = 3/4$. The amplitude can be represented as a vector $\psi_0$ with $N_D + 1$ components, $\psi_0 = (\Psi_0(x_0), \Psi_0(x_1), \ldots, \Psi_0(x_{N_D}))$. We recommend using complex NumPy arrays, e.g. `np.array([1+2j, 3+4j])` and `np.zeros((10, 10), dtype=np.complex64))`.

[1]:
```python
import matplotlib.pyplot as plt
import numpy as np
```

### 1.1 Problem 1A

Calculate the propagator matrix $\mathcal{K}_{8\epsilon}$ for a time period $T_0/16 = 8\epsilon$ (8 time steps) built from the elementary propagator matrix $\mathcal{K}_\epsilon$ for a single $\epsilon = \Delta t = T_0/128$ time step.

Recall we gave the general form of the propagator in lecture as

$$\mathcal{K}(x_b, t_b; x_a, t_a) = \left(\frac{m\omega}{2\pi i\hbar\, \sin(\omega(t_b - t_a))}\right)^{1/2} \exp\left(\frac{im\omega}{2\hbar\, \sin(\omega(t_b - t_a))}[(x_a^2 + x_b^2)\cos(\omega(t_b - t_a)) - 2x_a x_b]\right)$$

where $x_a$ and $x_b$ are the initial and final positions, respectively, and $t_a$ and $t_b$ are the initial and final times, respectively.

Use NumPy to print the matrix (default truncated output) and copy the (truncated) output into your report. Note by default if `K_8eps` is a large NumPy array, `print(K_8eps)` prints the first 3 and last 3 elements along each axis.

***Hint:*** *The elementary propagator matrix $\mathcal{K}_\epsilon$ is an $(N_D+1)\times(N_D+1)$-dimensional complex matrix that time evolves the state $\psi$ by one time step, and*

$$\mathcal{K}_t = (\Delta x)^{N-1}\mathcal{K}_\epsilon^N$$

*time evolves the state by $N$ time steps, where $N\epsilon = t$.*

### 1.2 Problem 1A Solution

In our problem, we can write

$$\mathcal{K}_\epsilon(x_b, x_a) = K(x_b, \epsilon, x_a, 0) = \sqrt{\frac{1}{2\pi i\hbar\ \sin\epsilon}}\exp\left(\frac{i}{2\hbar\ \sin\epsilon}[(x_a^2 + x_b^2)\ \cos\epsilon - 2x_a x_b]\right)$$

And by writing

$$(\mathcal{K}_\epsilon)_{i,j} = \mathcal{K}_\epsilon(x_i, x_j)$$

we can get matrix $\mathcal{K}$ at any time $t$.

[2]:
```
T0 = 2 * np.pi
NT = 128
DELTAT = T0 / NT
BOXSIZE = 8
ND = 600
DELTAX = BOXSIZE / ND
HBAR = 1

x = np.linspace(-BOXSIZE / 2, BOXSIZE / 2, ND + 1)

def func_K(x_a, x_b, dt):
    # exact analytical expression for the propgator
    coefficient = np.sqrt(1 / (2 * np.pi * 1j * HBAR * np.sin(dt)))
    exponent1 = 1j / (2 * HBAR * np.sin(dt))
    exponent2 = (x_a**2 + x_b**2) * np.cos(dt) - 2 * x_a * x_b
    return coefficient * np.exp(exponent1 * exponent2)
    # approximate expression for the propagator assuming dt is small
    # exponent = 1j * (0.5 * (x_b - x_a)**2 / dt - 0.5 * ((x_b + x_a) / 2)**2 *
  ↪dt)
    # return np.exp(exponent)

K_dt = np.zeros((ND + 1, ND + 1), dtype=np.complex64)
for i in range(ND + 1):
    for j in range(ND + 1):
```

```
        K_dt[i, j] = func_K(x[i], x[j], DELTAT)

K_8dt = DELTAX ** 7 * np.linalg.matrix_power(K_dt, 8)

print(K_8dt)
```

```
[[-0.05955856+0.01786728j -0.07576101+0.02170273j -0.07967375+0.05414084j
  … -0.46506009-0.30932614j -0.35439312-0.42735562j
  -0.21687411-0.50917214j]
 [-0.07576097+0.02170271j -0.08960781+0.03330221j -0.08308472+0.06724785j
  … -0.5340412 -0.16837658j -0.46141952-0.31131616j
  -0.3543929 -0.4273558j ]
 [-0.0796736 +0.05414084j -0.08308458+0.06724782j -0.06226945+0.09303052j
  … -0.5640738 -0.01040932j -0.5340413 -0.16837655j
  -0.4650599 -0.30932623j]
 …
 [-0.46505973-0.30932632j -0.5340409 -0.1683768j  -0.56407356-0.01040965j
  … -0.06226999+0.0930305j  -0.0830856 +0.06724777j
  -0.07967453+0.05413996j]
 [-0.35439283-0.42735574j -0.4614198 -0.31131664j -0.5340417 -0.16837694j
  … -0.08308599+0.06724777j -0.0896079 +0.03330208j
  -0.07576197+0.02170254j]
 [-0.21687397-0.5091721j  -0.3543929 -0.42735568j -0.46505997-0.3093263j
  … -0.07967461+0.0541404j  -0.07576212+0.02170259j
  -0.05955978+0.01786643j]]
```

# 2 Problem 1B

Evolve the probability amplitude of the electron *for a time period* $T_0/16 = 8\epsilon$ *(8 time steps)* and measure its mean position $\langle x \rangle$ as a function of time. Make a graph showing $\langle x \rangle$ versus time $t$. Label the axes.

***Hint:*** *Recall*

$$\langle x \rangle = \int x P_t(x) dx$$

*where* $P_t(x) = |\Psi_t(x)|^2$ *is the probability density function at time* $t$.

## 2.1 Problem 1B Solution

[3]:
```
XSTART = 0.75
ALPHA = 2

t = np.linspace(0, T0, NT + 1)

def func_psi_0(x, x_start):
    part1 = (ALPHA / np.pi) ** (1 / 4)
    part2 = np.exp(-ALPHA / 2 * (x - x_start)**2)
    return part1 * part2
```

```
psi_0 = func_psi_0(x, XSTART)
```

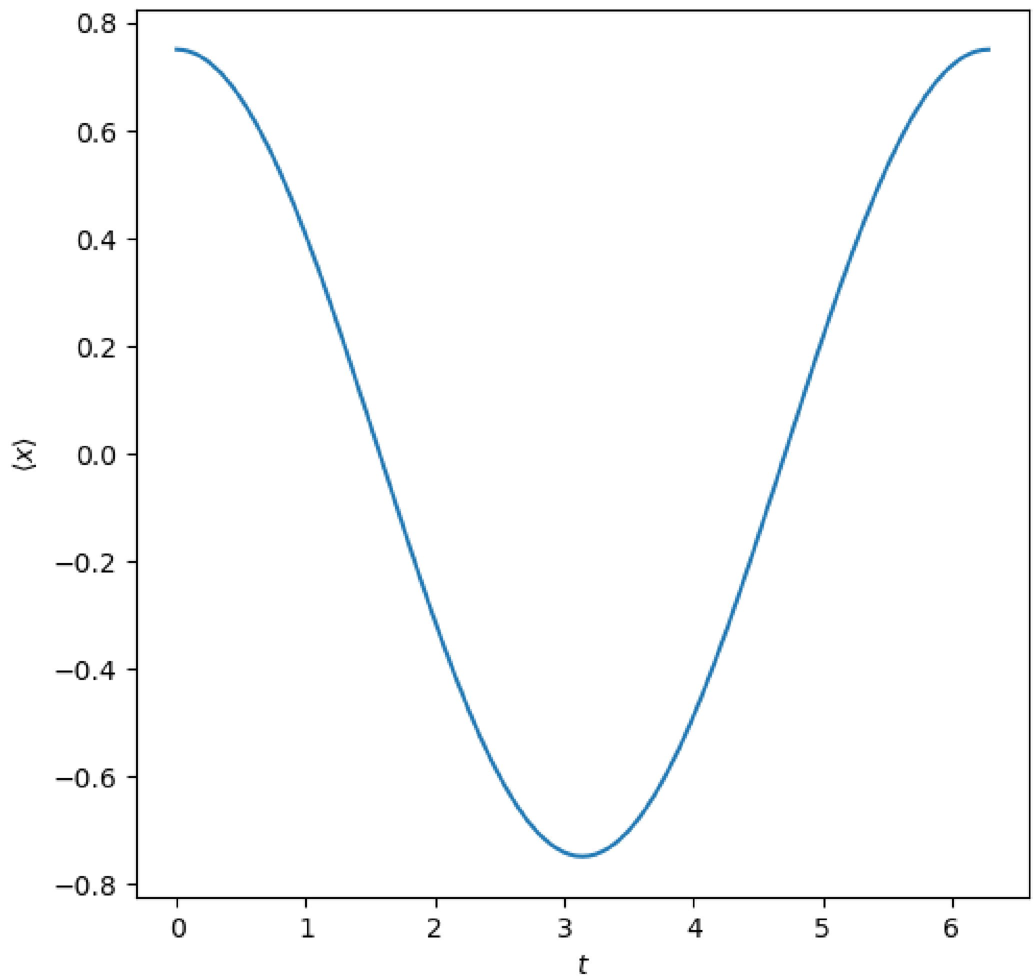
[4]:
```
psi = [psi_0]

for i in range(1, NT + 1):
    psi_t = DELTAX * np.matmul(K_dt, psi[i-1])
    psi_t /= np.sqrt(DELTAX * np.sum(psi_t * psi_t.conjugate()))
    psi.append(psi_t)

prob = []
for i in range(NT + 1):
    prob.append(np.real(psi[i] * psi[i].conjugate()))

x_bar = np.zeros_like(t)
for i in range(NT + 1):
    x_bar[i] = sum(prob[i] * x * DELTAX)

plt.figure(figsize=(6, 6))
plt.plot(t, x_bar)
plt.xlabel(r"$t$")
plt.ylabel(r"$\langle x \rangle$")
plt.show()
```




## 3 Problem 1C

Calculate the mean energy $\langle E \rangle$, mean kinetic energy $\langle K \rangle$, and mean potential energy $\langle V \rangle$ as a function of time. Make one graph showing all three with a legend labeling them.

***Hint:*** *Recall $E = K + V$ and for the mean value of $V$, we have*

$$\langle V \rangle = \int \frac{1}{2} x^2 P_t(x) dx.$$

As discussed in lecture, we will use another form of $\langle K \rangle$ for now which is simpler, but assuming we know some quantum mechanics

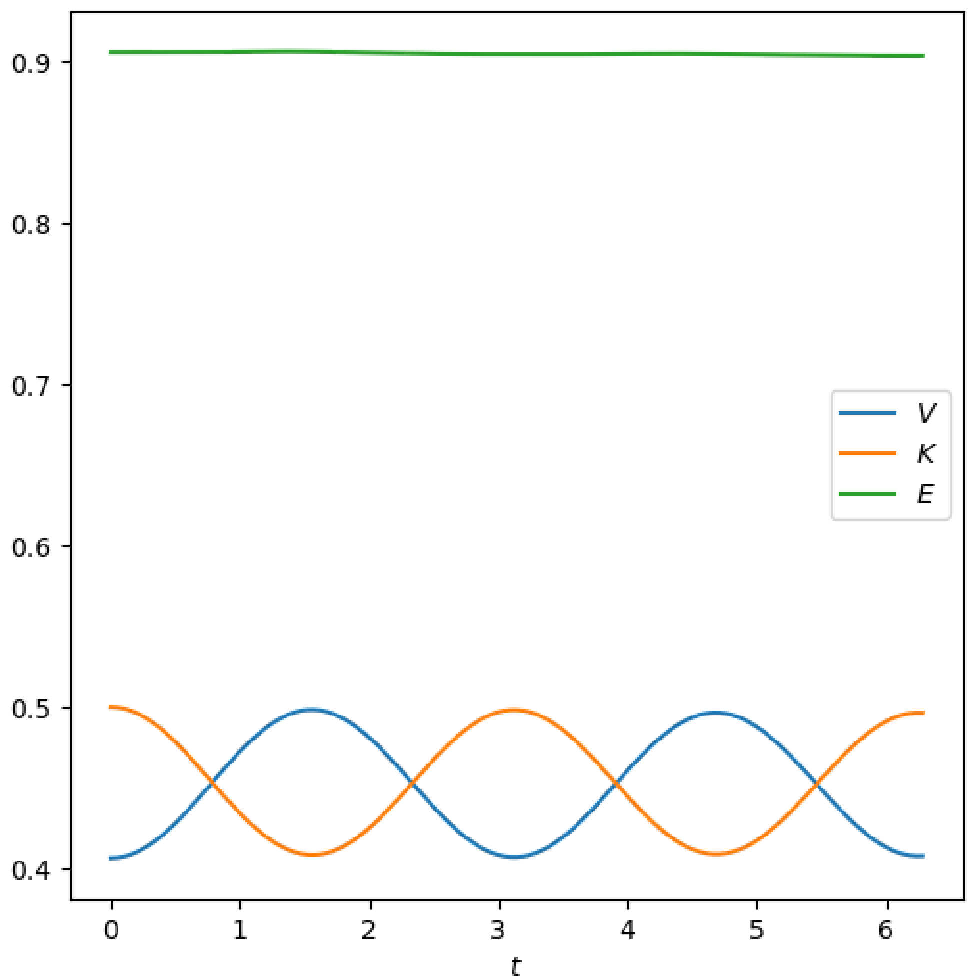
$$\langle K \rangle = \int \bar{\psi}(x) \left( -\frac{\hbar^2}{2} \frac{\partial^2}{\partial x^2} \right) \psi(x) dx = \frac{\hbar^2}{2} \int \left| \frac{\partial \psi}{\partial x} \right|^2 dx$$

[5]:

```
pot_en, kin_en, tot_en = [], [], []

for i in range(0, NT + 1):
    pot_en.append(sum(x**2 / 2 * prob[i] * DELTAX))
    psi_t = psi[i]
    dpsidx = (psi_t[2:] - psi_t[:-2]) / (2 * DELTAX)
    dpsidx2 = np.real(dpsidx * dpsidx.conjugate())
    kin_en.append(sum(HBAR**2 / 2 * dpsidx2 * DELTAX))
    tot_en.append(pot_en[-1] + kin_en[-1])


plt.figure(figsize=(6, 6))
plt.plot(t, pot_en, label=r"$V$")
plt.plot(t, kin_en, label=r'$K$')
plt.plot(t, tot_en, label=r"$E$")
plt.xlabel(r"$t$")
plt.legend()
plt.show()
```

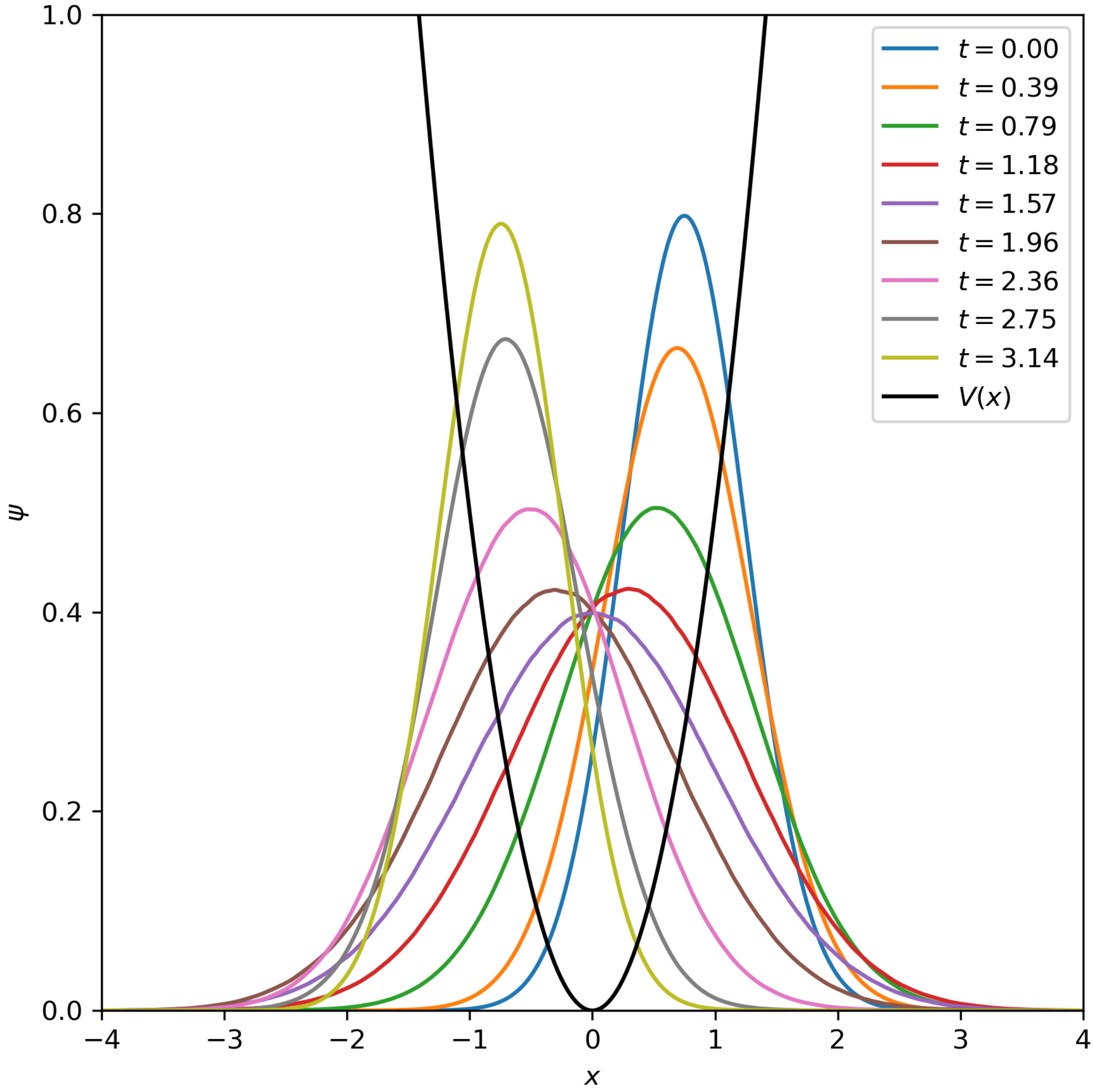


### 3.1 Problem 1D

Calculate the time evolution of the probability amplitude at times $mT_0/16$ for $m = 1, \ldots, 8$. Make a single graph showing the probability amplitudes initially and at those eight times, with a legend labeling them. Optional: Superimpose the potential $V(x)$ and amplitudes at those eight times on the same graph as the probability density function $P_t(x) = |\Psi_t(x)|^2$ at those eight times.

### 3.2 Problem 1D Solution

[6]:
```
plot_interval = 8
xmin, xmax, ymin, ymax = -BOXSIZE/2, BOXSIZE/2, 0, 1
plt.figure(dpi=300, figsize=(6, 6))
for i in range(0, NT // 2 + 1, plot_interval):
    plt.plot(x, prob[i], label=f"$t = {i * DELTAT:.2f}$")
plt.plot(x, x**2 / 2, 'k', label='$V(x)$')
plt.xlabel("$x$")
plt.ylabel("$\psi$")
plt.xlim([xmin, xmax])
plt.ylim([ymin, ymax])
plt.legend(loc='upper right')
plt.tight_layout()
plt.show()
```




### 3.3 Problem 1E

Animate the time evolution of the probability amplitude over the full time period $T_0 = 2\pi$. Each frame should correspond to one time step of $\epsilon = T_0/128$ (so $128 + 1$ frames total). Save the animation as a `.gif` or `.mp4` file.

### 3.4 Problem 1E Solutions

[7]:
```
!mkdir -p build

xmin, xmax, ymin, ymax = -BOXSIZE/2, BOXSIZE/2, -1, 1

plt.ioff()
```

```python
for i in range(0, NT):
    fig = plt.figure(dpi=300, figsize=(6, 6))
    plt.plot(x, prob[i], label='$|\Psi_t(x)|^2$')
    plt.plot(x, psi[i].real, label='$\Re[\Psi_t(x)]$')
    plt.plot(x, psi[i].imag, label='$\Im[\Psi_t(x)]$')
    plt.plot(x, x**2 / 2, 'k', label='$V(x)$')
    plt.xlabel("$x$")
    plt.ylabel("$\psi$ or $|\psi|^2$")
    plt.xlim([xmin, xmax])
    plt.ylim([ymin, ymax])
    plt.legend(loc='upper right', title=f'$t = {i * DELTAT:.2f}$')
    plt.savefig(f"build/anim_{i:03d}.png")
    plt.close(fig)
```

Now we can use `ffmpeg` to make a movie from the images.

[8]:
```
!ffmpeg -i build/anim_%03d.png -r 25 -y -pix_fmt yuv420p build/anim.mp4
```

```
ffmpeg version 7.1 Copyright (c) 2000-2024 the FFmpeg developers
  built with Apple clang version 16.0.0 (clang-1600.0.26.4)
  configuration: --prefix=/usr/local/Cellar/ffmpeg/7.1_3 --enable-shared
--enable-pthreads --enable-version3 --cc=clang --host-cflags= --host-
ldflags='-Wl,-ld_classic' --enable-ffplay --enable-gnutls --enable-gpl --enable-
libaom --enable-libaribb24 --enable-libbluray --enable-libdav1d --enable-
libharfbuzz --enable-libjxl --enable-libmp3lame --enable-libopus --enable-
librav1e --enable-librist --enable-librubberband --enable-libsnappy --enable-
libsrt --enable-libssh --enable-libsvtav1 --enable-libtesseract --enable-
libtheora --enable-libvidstab --enable-libvmaf --enable-libvorbis --enable-
libvpx --enable-libwebp --enable-libx264 --enable-libx265 --enable-libxml2
--enable-libxvid --enable-lzma --enable-libfontconfig --enable-libfreetype
--enable-frei0r --enable-libass --enable-libopencore-amrnb --enable-libopencore-
amrwb --enable-libopenjpeg --enable-libspeex --enable-libsoxr --enable-libzmq
--enable-libzimg --disable-libjack --disable-indev=jack --enable-videotoolbox
--enable-audiotoolbox
  libavutil      59. 39.100 / 59. 39.100
  libavcodec     61. 19.100 / 61. 19.100
  libavformat    61.  7.100 / 61.  7.100
  libavdevice    61.  3.100 / 61.  3.100
  libavfilter    10.  4.100 / 10.  4.100
  libswscale      8.  3.100 /  8.  3.100
  libswresample   5.  3.100 /  5.  3.100
  libpostproc    58.  3.100 / 58.  3.100
Input #0, image2, from 'build/anim_%03d.png':
  Duration: 00:00:05.12, start: 0.000000, bitrate: N/A
  Stream #0:0: Video: png, rgba(pc, gbr/unknown/unknown), 1800x1800 [SAR
11811:11811 DAR 1:1], 25 fps, 25 tbr, 25 tbn
Stream mapping:
  Stream #0:0 -> #0:0 (png (native) -> h264 (libx264))
```

```
Press [q] to stop, [?] for help
[libx264 @ 0x7f91d9115dc0] using SAR=1/1
[libx264 @ 0x7f91d9115dc0] using cpu capabilities: MMX2 SSE2Fast
SSSE3 SSE4.2 AVX FMA3 BMI2 AVX2
[libx264 @ 0x7f91d9115dc0] profile High, level 5.0, 4:2:0, 8-bit
[libx264 @ 0x7f91d9115dc0] 264 - core 164 r3108 31e19f9 -
H.264/MPEG-4 AVC codec - Copyleft 2003-2023 - http://www.videolan.org/x264.html
- options: cabac=1 ref=3 deblock=1:0:0 analyse=0x3:0x113 me=hex subme=7 psy=1
psy_rd=1.00:0.00 mixed_ref=1 me_range=16 chroma_me=1 trellis=1 8x8dct=1 cqm=0
deadzone=21,11 fast_pskip=1 chroma_qp_offset=-2 threads=12 lookahead_threads=2
sliced_threads=0 nr=0 decimate=1 interlaced=0 bluray_compat=0
constrained_intra=0 bframes=3 b_pyramid=2 b_adapt=1 b_bias=0 direct=1 weightb=1
open_gop=0 weightp=2 keyint=250 keyint_min=25 scenecut=40 intra_refresh=0
rc_lookahead=40 rc=crf mbtree=1 crf=23.0 qcomp=0.60 qpmin=0 qpmax=69 qpstep=4
ip_ratio=1.40 aq=1:1.00
Output #0, mp4, to 'build/anim.mp4':
  Metadata:
    encoder         : Lavf61.7.100
  Stream #0:0: Video: h264 (avc1 / 0x31637661), yuv420p(tv, progressive),
1800x1800 [SAR 1:1 DAR 1:1], q=2-31, 25 fps, 12800 tbn
      Metadata:
        encoder         : Lavc61.19.100 libx264
      Side data:
        cpb: bitrate max/min/avg: 0/0/0 buffer size: 0 vbv_delay: N/A
[out#0/mp4 @ 0x7f91d90209c0] video:471KiB audio:0KiB subtitle:0KiB
other streams:0KiB global headers:0KiB muxing overhead: 0.469135%
frame=  128 fps= 33 q=-1.0 Lsize=     473KiB time=00:00:05.04 bitrate=
768.6kbits/s speed=1.31x
[libx264 @ 0x7f91d9115dc0] frame I:1     Avg QP:14.01  size: 37667
[libx264 @ 0x7f91d9115dc0] frame P:55    Avg QP:23.72  size:  5196
[libx264 @ 0x7f91d9115dc0] frame B:72    Avg QP:27.42  size:  2192
[libx264 @ 0x7f91d9115dc0] consecutive B-frames: 10.2% 34.4% 30.5%
25.0%
[libx264 @ 0x7f91d9115dc0] mb I  I16..4: 65.1% 28.2%  6.6%
[libx264 @ 0x7f91d9115dc0] mb P  I16..4:  0.3%  1.2%  0.3%  P16..4:
2.8%  1.2%  0.8%  0.0%  0.0%    skip:93.3%
[libx264 @ 0x7f91d9115dc0] mb B  I16..4:  0.1%  0.1%  0.0%  B16..8:
4.0%  1.3%  0.2%  direct: 0.0%  skip:94.3%  L0:49.8% L1:39.4% BI:10.8%
[libx264 @ 0x7f91d9115dc0] 8x8 transform intra:49.0% inter:31.4%
[libx264 @ 0x7f91d9115dc0] coded y,uvDC,uvAC intra: 7.8% 11.2% 10.1%
inter: 0.3% 0.7% 0.4%
[libx264 @ 0x7f91d9115dc0] i16 v,h,dc,p: 74% 24%  2%  0%
[libx264 @ 0x7f91d9115dc0] i8 v,h,dc,ddl,ddr,vr,hd,vl,hu: 20%  3% 75%
0%  0%  0%  0%  0%  0%
[libx264 @ 0x7f91d9115dc0] i4 v,h,dc,ddl,ddr,vr,hd,vl,hu: 17% 28% 33%
4%  4%  4%  4%  3%  4%
[libx264 @ 0x7f91d9115dc0] i8c dc,h,v,p: 83% 12%  5%  0%
[libx264 @ 0x7f91d9115dc0] Weighted P-Frames: Y:0.0% UV:0.0%
```

```
[libx264 @ 0x7f91d9115dc0] ref P L0: 71.0%  2.5% 17.1%  9.4%
[libx264 @ 0x7f91d9115dc0] ref B L0: 82.1% 15.3%  2.6%
[libx264 @ 0x7f91d9115dc0] ref B L1: 99.0%  1.0%
[libx264 @ 0x7f91d9115dc0] kb/s:751.97
```

[9]:
```python
# watch the video
import IPython.display as ipd
ipd.Video('build/anim.mp4')
```

[9]:
```
<IPython.core.display.Video object>
```

### 3.5 Bonus: Visualize propagator $\mathcal{K}_t$

[10]:
```python
K_t = [K_dt]
for i in range(1, NT):
    K_t.append(DELTAX * np.matmul(K_dt, K_t[i-1]))
K_t = np.array(K_t)
K_t_trace = np.trace(K_t, axis1=1, axis2=2)
```

[11]:
```python
plt.ioff()
for i in range(1, NT):
    plt.figure(dpi=300, figsize=(12, 6))
    plt.subplot(121)
    plt.imshow(K_t[i].real, cmap='Blues', origin='upper', extent=[-BOXSIZE/2,␣
↪BOXSIZE/2, -BOXSIZE/2, BOXSIZE/2])
    cbar1 = plt.colorbar()
    plt.title(f"$t = {i * DELTAT:.2f}$")
    plt.ylabel("$x$")
    plt.xlabel("$x'$")
    cbar1.set_label("$\Re[K_t(x, t; x', 0)]$")
    plt.subplot(122)
    plt.imshow(K_t[i].imag, cmap='Reds', origin='upper', extent=[-BOXSIZE/2,␣
↪BOXSIZE/2, -BOXSIZE/2, BOXSIZE/2])
    cbar2 = plt.colorbar()
    plt.title(f"$t = {i * DELTAT:.2f}$")
    plt.ylabel("$x$")
    plt.xlabel("$x'$")
    cbar2.set_label("$\Im[K_t(x, t; x', 0)]$")
    plt.savefig(f"build/propagator_{i:03d}.png")
```

```
/var/folders/75/5drbyjls2klg498kdn74p5_r0000gn/T/ipykernel_85161/2590942493.py:3
: RuntimeWarning: More than 20 figures have been opened. Figures created through
the pyplot interface (`matplotlib.pyplot.figure`) are retained until explicitly
closed and may consume too much memory. (To control this warning, see the
rcParam `figure.max_open_warning`). Consider using `matplotlib.pyplot.close()`.
  plt.figure(dpi=300, figsize=(12, 6))
```

[12]:
```
!ffmpeg -i build/propagator_%03d.png -r 25 -y -pix_fmt yuv420p build/propagator.
↪mp4
```

```
ffmpeg version 7.1 Copyright (c) 2000-2024 the FFmpeg developers
  built with Apple clang version 16.0.0 (clang-1600.0.26.4)
  configuration: --prefix=/usr/local/Cellar/ffmpeg/7.1_3 --enable-shared
--enable-pthreads --enable-version3 --cc=clang --host-cflags= --host-
ldflags='-Wl,-ld_classic' --enable-ffplay --enable-gnutls --enable-gpl --enable-
libaom --enable-libaribb24 --enable-libbluray --enable-libdav1d --enable-
libharfbuzz --enable-libjxl --enable-libmp3lame --enable-libopus --enable-
librav1e --enable-librist --enable-librubberband --enable-libsnappy --enable-
libsrt --enable-libssh --enable-libsvtav1 --enable-libtesseract --enable-
libtheora --enable-libvidstab --enable-libvmaf --enable-libvorbis --enable-
libvpx --enable-libwebp --enable-libx264 --enable-libx265 --enable-libxml2
--enable-libxvid --enable-lzma --enable-libfontconfig --enable-libfreetype
--enable-frei0r --enable-libass --enable-libopencore-amrnb --enable-libopencore-
amrwb --enable-libopenjpeg --enable-libspeex --enable-libsoxr --enable-libzmq
--enable-libzimg --disable-libjack --disable-indev=jack --enable-videotoolbox
--enable-audiotoolbox
  libavutil      59. 39.100 / 59. 39.100
  libavcodec     61. 19.100 / 61. 19.100
  libavformat    61.  7.100 / 61.  7.100
  libavdevice    61.  3.100 / 61.  3.100
  libavfilter    10.  4.100 / 10.  4.100
  libswscale      8.  3.100 /  8.  3.100
  libswresample   5.  3.100 /  5.  3.100
  libpostproc    58.  3.100 / 58.  3.100
Input #0, image2, from 'build/propagator_%03d.png':
  Duration: 00:00:05.08, start: 0.000000, bitrate: N/A
  Stream #0:0: Video: png, rgba(pc, gbr/unknown/unknown), 3600x1800 [SAR
11811:11811 DAR 2:1], 25 fps, 25 tbr, 25 tbn
Stream mapping:
  Stream #0:0 -> #0:0 (png (native) -> h264 (libx264))
Press [q] to stop, [?] for help
[libx264 @ 0x7f7d72f14200] using SAR=1/1
[libx264 @ 0x7f7d72f14200] using cpu capabilities: MMX2 SSE2Fast
SSSE3 SSE4.2 AVX FMA3 BMI2 AVX2
[libx264 @ 0x7f7d72f14200] profile High, level 5.1, 4:2:0, 8-bit
[libx264 @ 0x7f7d72f14200] 264 - core 164 r3108 31e19f9 -
H.264/MPEG-4 AVC codec - Copyleft 2003-2023 - http://www.videolan.org/x264.html
- options: cabac=1 ref=3 deblock=1:0:0 analyse=0x3:0x113 me=hex subme=7 psy=1
psy_rd=1.00:0.00 mixed_ref=1 me_range=16 chroma_me=1 trellis=1 8x8dct=1 cqm=0
deadzone=21,11 fast_pskip=1 chroma_qp_offset=-2 threads=12 lookahead_threads=2
sliced_threads=0 nr=0 decimate=1 interlaced=0 bluray_compat=0
constrained_intra=0 bframes=3 b_pyramid=2 b_adapt=1 b_bias=0 direct=1 weightb=1
open_gop=0 weightp=2 keyint=250 keyint_min=25 scenecut=40 intra_refresh=0
rc_lookahead=40 rc=crf mbtree=1 crf=23.0 qcomp=0.60 qpmin=0 qpmax=69 qpstep=4
ip_ratio=1.40 aq=1:1.00
```

```
Output #0, mp4, to 'build/propagator.mp4':
  Metadata:
    encoder         : Lavf61.7.100
  Stream #0:0: Video: h264 (avc1 / 0x31637661), yuv420p(tv, progressive),
3600x1800 [SAR 1:1 DAR 2:1], q=2-31, 25 fps, 12800 tbn
      Metadata:
        encoder         : Lavc61.19.100 libx264
      Side data:
        cpb: bitrate max/min/avg: 0/0/0 buffer size: 0 vbv_delay: N/A
[out#0/mp4 @ 0x7f7d72f132c0] video:4302KiB audio:0KiB subtitle:0KiB
other streams:0KiB global headers:0KiB muxing overhead: 0.054684%
frame=  127 fps= 10 q=-1.0 Lsize=    4304KiB time=00:00:05.00
bitrate=7052.3kbits/s speed=0.407x
[libx264 @ 0x7f7d72f14200] frame I:1     Avg QP:21.48  size:128256
[libx264 @ 0x7f7d72f14200] frame P:37    Avg QP:23.56  size: 40785
[libx264 @ 0x7f7d72f14200] frame B:89    Avg QP:26.41  size: 31093
[libx264 @ 0x7f7d72f14200] consecutive B-frames:  2.4%  7.9% 14.2%
75.6%
[libx264 @ 0x7f7d72f14200] mb I  I16..4: 31.8% 53.7% 14.5%
[libx264 @ 0x7f7d72f14200] mb P  I16..4:  9.1% 10.8%  0.7%  P16..4:
10.6%  2.9%  0.4%  0.0%  0.0%    skip:65.6%
[libx264 @ 0x7f7d72f14200] mb B  I16..4:  3.4%  2.9%  0.2%  B16..8:
14.9%  5.4%  0.4%  direct: 1.2%  skip:71.5%  L0:50.0% L1:43.0% BI: 7.0%
[libx264 @ 0x7f7d72f14200] 8x8 transform intra:49.6% inter:94.8%
[libx264 @ 0x7f7d72f14200] coded y,uvDC,uvAC intra: 25.6% 62.4% 9.9%
inter: 6.5% 4.5% 0.0%
[libx264 @ 0x7f7d72f14200] i16 v,h,dc,p: 12% 10%  1% 77%
[libx264 @ 0x7f7d72f14200] i8 v,h,dc,ddl,ddr,vr,hd,vl,hu: 24% 16% 14%
5% 14%  8%  8%  7%  4%
[libx264 @ 0x7f7d72f14200] i4 v,h,dc,ddl,ddr,vr,hd,vl,hu: 25% 15% 14%
2% 34%  3%  3%  3%  1%
[libx264 @ 0x7f7d72f14200] i8c dc,h,v,p: 39% 17% 17% 27%
[libx264 @ 0x7f7d72f14200] Weighted P-Frames: Y:0.0% UV:0.0%
[libx264 @ 0x7f7d72f14200] ref P L0: 53.5%  4.1% 27.7% 14.6%
[libx264 @ 0x7f7d72f14200] ref B L0: 72.2% 19.3%  8.6%
[libx264 @ 0x7f7d72f14200] ref B L1: 92.2%  7.8%
[libx264 @ 0x7f7d72f14200] kb/s:6936.37
```

[13]:
```python
# watch the video
import IPython.display as ipd
ipd.Video('build/propagator.mp4')
```

[13]: <IPython.core.display.Video object>

[ ]: