# Ph138A HW2

1. a. $$\rho(\varepsilon) = \frac{2|\varepsilon|}{W^2}\Theta(W-\varepsilon)$$

Hence $$\nu(\varepsilon) = \int_0^{\varepsilon} \frac{2\varepsilon'}{W^2}d\varepsilon' = \frac{\varepsilon^2}{W^2} \Rightarrow \varepsilon = W\sqrt{\nu}.$$

$$E_k(\varepsilon) = \int \rho(\varepsilon')\varepsilon' d\varepsilon' = \frac{2\varepsilon^3}{3W^2} = \frac{2}{3}W|\nu|^{3/2}.$$

b. $$W\,\mathrm{sgn}(\nu_\alpha)\sqrt{|\nu_\alpha|} + U(\nu-\nu_\alpha) = \mu$$

we'll have phase transition

at $\frac{\mu}{W} \sim 0.80$, $1.15$, $1.45$

and plot below:

c. also see plot below.

d. It shifts to larger U.

Because lower DOS → weaker Stoner strike

→ bigger U to reach $\rho U \approx 1$.

2. a. Moiré period comes from their near-cancellation, so slight mismatch is magnified.

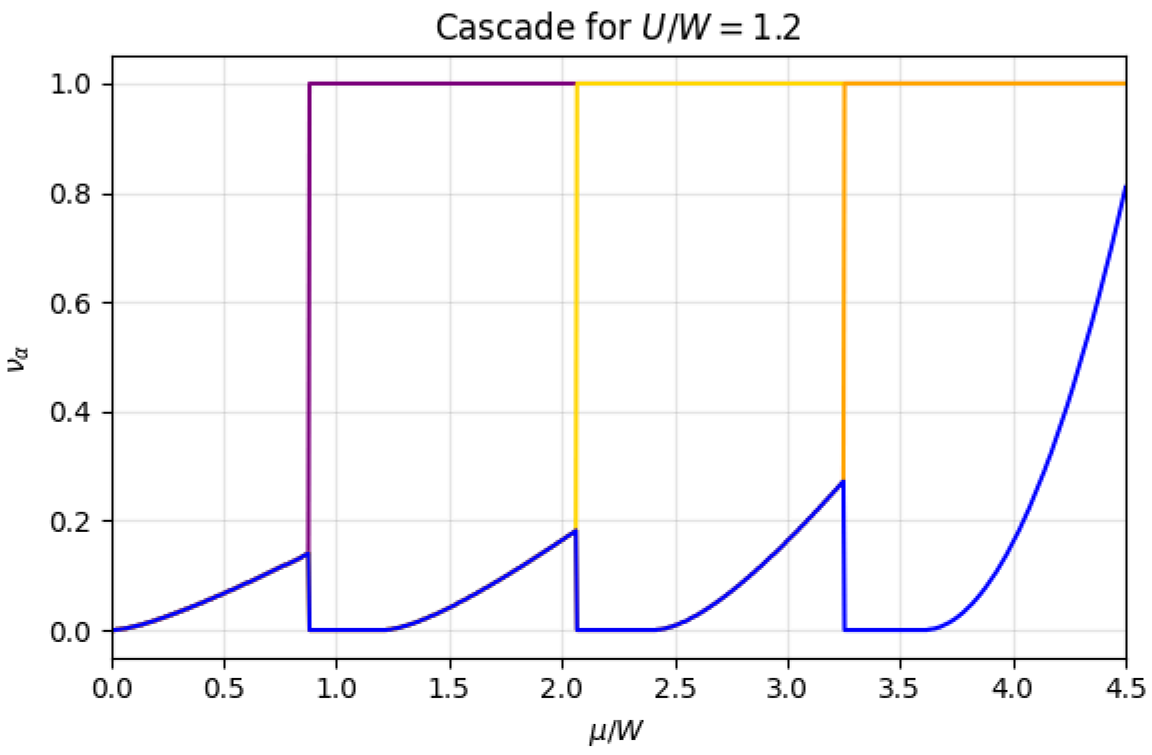
b. see below

c. see below

## 1.(b)

In [12]:
```
"""
Cascade plot for P 1 b (linear DOS, U/W = 1.2).

Produces the ν₁…ν₄ vs μ/W figure shown earlier.
Requires: numpy, matplotlib
"""

import numpy as np
import matplotlib.pyplot as plt

# --- parameters ---------------------------------------------------------------
W = 1.0           # set bandwidth units
U = 1.2 * W       # interaction
mu_vals = np.linspace(0, 4.5*W, 900)   # chemical-potential sweep
n_grid = np.linspace(0.0, 1.0, 1001)   # partial-filling grid

# --- helper: total mean-field energy --------------------------------------
def energy(nu_vec, mu, W=1.0, U=1.2):
    ν = np.asarray(nu_vec)
    kinetic = (2/3) * W * np.sum(ν**1.5)
    hartree = 0.5 * U * ((ν.sum())**2 - np.sum(ν**2))
    chem = -mu * ν.sum()
    return kinetic + hartree + chem

# --- candidate flavour patterns ---------------------------------------------
patterns = [
    lambda n: np.full(4, n),              # (4,0)  all equal
    lambda n: np.array([1.0, n, n, n]),   # (3,1)
    lambda n: np.array([1.0, 1.0, n, n]),# (2,2)
    lambda n: np.array([1.0, 1.0, 1.0,n]),# (1,3)
    lambda n: np.ones(4)                  # (0,4)
]

# --- ground-state search ------------------------------------------------------
nu_alpha = np.zeros((len(mu_vals), 4))

for i_mu, mu in enumerate(mu_vals):
    best_E, best_vec = np.inf, None
    for pat in patterns:
        for n in n_grid:
            vec = pat(n)
            if np.any(vec > 1):           # skip over-filled
                continue
            E = energy(vec, mu, W, U)
            if E < best_E:
                best_E, best_vec = E, vec
    nu_alpha[i_mu] = np.sort(best_vec)[::-1]   # sort for colour order

# --- plot ---------------------------------------------------------------------
cols = ['purple', 'gold', 'orange', 'blue']
plt.figure(figsize=(6,4))
```

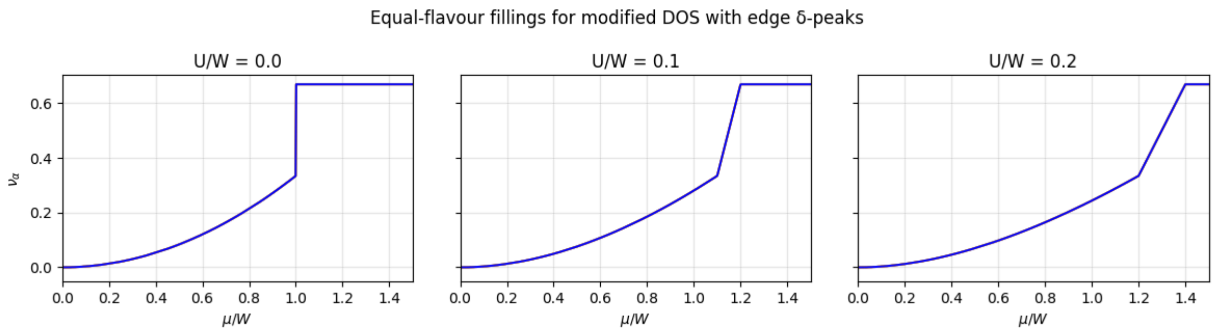
```
for j in range(4):
    plt.plot(mu_vals/W, nu_alpha[:, j], lw=1.5, color=cols[j])
plt.xlabel(r'$\mu/W$');  plt.ylabel(r'$\nu_\alpha$')
plt.title(r'Cascade for $U/W=1.2$')
plt.xlim(0, 4.5); plt.ylim(-0.05, 1.05)
plt.grid(alpha=0.3); plt.tight_layout()
plt.show()
```



## 1.(c)

In [13]:

```
"""
Plot for P 1 c – modified DOS with edge δ-peaks.
Three panels (U/W = 0, 0.1, 0.2), equal-flavour solution.

Needs: numpy, matplotlib
"""

import numpy as np
import matplotlib.pyplot as plt

# –– constants ––––––––––––––––––––––––––––––––––––––––––––––––––––––––––
W = 1.0                           # set bandwidth units
Uvals = [0.0, 0.1*W, 0.2*W]       # three interaction strengths
mu = np.linspace(0, 1.5*W, 600)  # chemical–potential sweep
nu_max = 2/3                      # single–flavour filling limit
n_grid = np.linspace(0, nu_max, 2001)

nu = 0.165                        # Poisson ratio (appears only in part 2)

# –– single–flavour kinetic energy & ε(ν) for modified DOS ––––––––––––––––
def Ek_single(n):
    n = np.asarray(n)
    Ek = np.empty_like(n)
    mask = n < 1/3
    Ek[mask]  = (2/3)*np.sqrt(3)*W*n[mask]**1.5
    Ek[~mask] = (2/9)*W + W*(n[~mask] - 1/3)
    return Ek

# pre-tabulate for speed
Ek_tab = Ek_single(n_grid)

# —— figure ---------------------------------------------------------------
fig, axs = plt.subplots(1, 3, figsize=(12, 3.3), sharey=True)
colors = ['purple', 'gold', 'orange', 'blue']

for ax, U in zip(axs, Uvals):
    νμ = np.zeros((len(mu), 4))          # να(μ)

    # pre-compute Hartree energy per n for equal fillings: ν_tot = 4n
    ν_tot = 4*n_grid
    H_tab = 0.5*U*(ν_tot**2 - 4*n_grid**2)

    for i, m in enumerate(mu):
        # energy per trial filling n (all equal)
        E = 4*Ek_tab + H_tab - m*ν_tot
        n_opt = n_grid[np.argmin(E)]
        νμ[i] = np.full(4, n_opt)

    for j in range(4):
        ax.plot(mu/W, νμ[:, j], color=colors[j], lw=1.4)

    ax.set_title(f'U/W = {U/W:.1f}')
    ax.set_xlabel(r'$\mu/W$')
    ax.set_xlim(0, 1.5);  ax.set_ylim(-0.05, 0.7)
    ax.grid(alpha=0.3)
axs[0].set_ylabel(r'$\nu_\alpha$')
fig.suptitle('Equal-flavour fillings for modified DOS with edge δ-peaks')
plt.tight_layout()
plt.show()
```



## 2.(b)

In [14]:
```
import numpy as np
from scipy.optimize import least_squares
```

```
# --- constants ----------------------------------------------------------------
a0 = 0.246   # graphene lattice constant [nm]
k_mag = 4*np.pi/(np.sqrt(3)*a0)           # |K| in the unstrained layer
nu = 0.165   # Poisson ratio

# unstrained reciprocal-lattice vectors of layer-2 (columns)
k_vecs = np.array([[ 1.0,             -0.5,        -0.5],
                   [ 0.0,  np.sqrt(3)/2, -np.sqrt(3)/2]]) * k_mag

# target moiré periods [nm]
L_target = np.array([13.72, 12.70, 10.18])

# --- helpers ------------------------------------------------------------------
def R(theta):
    c, s = np.cos(theta), np.sin(theta)
    return np.array([[c, -s],
                     [s,  c]])

def strain_matrix(eps, phi):
    sx = 1.0 / (1.0 + eps)          # reciprocal-space scaling ∥
    sy = 1.0 / (1.0 - nu*eps)       # reciprocal-space scaling ⊥
    return R(phi) @ np.diag([sx, sy]) @ R(-phi)

def moire_lengths(theta, eps, phi):
    K_diff = R(theta) @ k_vecs - strain_matrix(eps, phi) @ k_vecs
    K_len = np.linalg.norm(K_diff, axis=0)
    return 4*np.pi/(np.sqrt(3)*K_len)

def residual(params):
    theta, eps, phi = params
    return moire_lengths(theta, eps, phi) - L_target

# --- initial guess ------------------------------------------------------------
p0 = np.array([np.deg2rad(1.15), 0.00685, np.deg2rad(25.0)])

print("Initial guess parameters:")
print(f"  θ = {np.rad2deg(p0[0]):.3f} °,  ε = {100*p0[1]:.3f} %,  φ = {np.ra
print("Initial moiré periods (nm):", moire_lengths(*p0))

# --- least-squares fit with verbose output ------------------------------------
res = least_squares(residual, p0,
                    bounds=([np.deg2rad(0.5), 0.0,      0.0],
                            [np.deg2rad(2.0), 0.03, np.deg2rad(60)]),
                    verbose=2)

theta_fit, eps_fit, phi_fit = res.x
L_fit = moire_lengths(theta_fit, eps_fit, phi_fit)

# --- results ------------------------------------------------------------------
print("\nOptimisation finished.")
print("fitted parameters:")
print(f"  twist angle θ     = {np.rad2deg(theta_fit):.4f} °")
print(f"  heterostrain ε    = {100*eps_fit:.4f} %")
print(f"  strain axis φ     = {np.rad2deg(phi_fit):.2f} °")
print("\nreproduced moiré periods:")
for i, L in enumerate(L_fit, 1):
    print(f"  |M_{i}| = {L:.4f} nm  (target {L_target[i-1]:.2f} nm)")

print("\nFull optimisation report:")
print(res)
```

```
Initial guess parameters:
  θ = 1.150 °,  ε = 0.685 %,  φ = 25.00 °
Initial moiré periods (nm): [10.38690253 11.82814161 14.62358618]
   Iteration     Total nfev        Cost      Cost reduction    Step norm     
Optimality
       0              1         1.5808e+01                                    
2.70e+01
       1              2         6.8864e+00      8.92e+00       1.61e-01       
3.99e+00
       2              3         3.8830e+00      3.00e+00       2.37e-01       
2.79e+00
       3              4         3.3519e+00      5.31e-01       2.01e-01       
5.46e-01
       4              5         3.3216e+00      3.03e-02       1.24e-02       
3.27e-02
       5              6         3.3204e+00      1.17e-03       5.37e-06       
1.47e-03
       6              7         3.3204e+00      8.33e-06       1.21e-07       
9.22e-06
       7              8         3.3204e+00      1.11e-10       7.34e-10       
1.09e-06
Both `ftol` and `xtol` termination conditions are satisfied.
Function evaluations 8, initial cost 1.5808e+01, final cost 3.3204e+00, firs
t-order optimality 1.09e-06.

Optimisation finished.
fitted parameters:
  twist angle θ     = 1.1553 °
  heterostrain ε    = 0.0000 %
  strain axis φ     = 60.00 °

reproduced moiré periods:
  |M_1| = 12.2000 nm  (target 13.72 nm)
  |M_2| = 12.2000 nm  (target 12.70 nm)
  |M_3| = 12.2000 nm  (target 10.18 nm)

Full optimisation report:
     message: Both `ftol` and `xtol` termination conditions are satisfied.
     success: True
      status: 4
         fun: [-1.520e+00 -5.000e-01  2.020e+00]
           x: [ 2.016e-02  3.172e-18  1.047e+00]
        cost: 3.3204000000000304
         jac: [[-6.050e+02 -3.044e+02 -6.830e-07]
               [-6.050e+02  3.060e+02 -0.000e+00]
               [-6.050e+02  6.100e+00 -0.000e+00]]
        grad: [ 2.906e-06  3.221e+02  1.038e-06]
  optimality: 1.0871887238137105e-06
 active_mask: [ 0 -1  1]
        nfev: 8
        njev: 8
```

## 2.(c)

In [15]:
```
"""
Fit θ (twist), ε (heterostrain) and φ (strain axis) to
L_target = {13.0, 13.0, 12.0} nm   (P 2 c).

Requires: numpy, scipy   ▸  pip install numpy scipy
"""

import numpy as np
from scipy.optimize import least_squares

# --- constants ------------------------------------------------------------
a0 = 0.246                        # graphene lattice constant [nm]
k_mag = 4*np.pi/(np.sqrt(3)*a0)  # |K| of the unstrained layer
nu = 0.165                        # Poisson ratio

# layer-2 reciprocal-lattice basis (columns)
k_vecs = np.array([[ 1.0,           -0.5,         -0.5],
                   [ 0.0, np.sqrt(3)/2, -np.sqrt(3)/2]]) * k_mag

# target moiré periods [nm]
L_target = np.array([13.0, 13.0, 12.0])

# --- helpers --------------------------------------------------------------
def R(theta):
    c, s = np.cos(theta), np.sin(theta)
    return np.array([[c, -s],
                     [s,  c]])

def strain_matrix(eps, phi):
    sx = 1.0 / (1.0 + eps)        # reciprocal scaling ∥ to strain
    sy = 1.0 / (1.0 - nu*eps)     # reciprocal scaling ⊥
    return R(phi) @ np.diag([sx, sy]) @ R(-phi)

def moire_lengths(theta, eps, phi):
    ΔK = R(theta) @ k_vecs - strain_matrix(eps, phi) @ k_vecs
    K_len = np.linalg.norm(ΔK, axis=0)
    return 4*np.pi/(np.sqrt(3)*K_len)    # real-space periods

def residual(p):
    θ, ε, φ = p
    return moire_lengths(θ, ε, φ) - L_target

# --- initial guess --------------------------------------------------------
p0 = np.array([np.deg2rad(1.10), 0.0027, 0.0])  # θ≈1.10°, ε≈0.27 %, φ=0°

print("Initial guess:")
print(f"  θ = {np.rad2deg(p0[0]):.3f} °,  ε = {100*p0[1]:.3f} %,  φ = {np.ra
print("Initial moiré periods (nm):", moire_lengths(*p0))

# --- least-squares fit (verbose) ------------------------------------------
res = least_squares(
        residual, p0,
        bounds=([np.deg2rad(0.5), 0.0,      0.0],
                [np.deg2rad(2.0), 0.03, np.deg2rad(60)]),
        verbose=2
)

θ_fit, ε_fit, φ_fit = res.x
L_fit = moire_lengths(θ_fit, ε_fit, φ_fit)

# --- results ---------------------------------------------------------
print("\nOptimisation finished.")
print(f"  twist angle θ  = {np.rad2deg(θ_fit):.4f} °")
print(f"  heterostrain ε = {100*ε_fit:.4f} %")
print(f"  strain axis φ  = {np.rad2deg(φ_fit):.2f} °")
print("\nReproduced moiré periods:")
for i, L in enumerate(L_fit, 1):
    print(f"  |M_{i}| = {L:.4f} nm  (target {L_target[i-1]:.2f} nm)")

print("\nFull optimisation report:")
print(res)
```

```
Initial guess:
  θ = 1.100 °,  ε = 0.270 %,  φ = 0.00 °
Initial moiré periods (nm): [12.70619673 11.96672482 13.78990766]
   Iteration     Total nfev        Cost      Cost reduction    Step norm
Optimality
       0              1         2.1789e+00
9.29e+00
       1              2         8.3290e-01      1.35e+00       1.64e-03
8.22e-01
       2              3         4.4812e-01      3.85e-01       7.86e-04
3.57e-01
       3              4         3.4846e-01      9.97e-02       2.71e-04
1.02e-01
       4              5         3.3375e-01      1.47e-02       4.48e-05
1.60e-02
       5              6         3.3334e-01      4.16e-04       1.73e-06
5.39e-04
       6              7         3.3333e-01      2.43e-06       1.66e-01
3.94e-06
       7              8         3.3333e-01      1.56e-11       3.04e-01
2.23e-06
       8              9         3.3333e-01      1.10e-14       5.54e-02
5.38e-07
       9             11         3.3333e-01      1.15e-13       4.02e-03
1.96e-06
      10             21         3.3333e-01      0.00e+00       0.00e+00
1.96e-06
`xtol` termination condition is satisfied.
Function evaluations 21, initial cost 2.1789e+00, final cost 3.3333e-01, fir
st-order optimality 1.96e-06.

Optimisation finished.
  twist angle θ  = 1.1128 °
  heterostrain ε = 0.0000 %
  strain axis φ  = 23.94 °

Reproduced moiré periods:
  |M_1| = 12.6667 nm  (target 13.00 nm)
  |M_2| = 12.6667 nm  (target 13.00 nm)
  |M_3| = 12.6667 nm  (target 12.00 nm)

Full optimisation report:
     message: `xtol` termination condition is satisfied.
     success: True
      status: 3
         fun: [-3.333e-01 -3.333e-01  6.667e-01]
           x: [ 1.942e-02  7.773e-32  4.179e-01]
        cost: 0.333333333333238
         jac: [[-6.522e+02 -2.767e+02 -4.768e-07]
               [-6.522e+02 -8.066e+01 -4.530e-06]
               [-6.522e+02  3.653e+02  4.530e-06]]
        grad: [ 4.400e-06  3.627e+02  4.689e-06]
  optimality: 1.9595598005295266e-06
 active_mask: [ 0 -1  0]
        nfev: 21
        njev: 10
```