Disclosure

AI used to help generate plots & figures

Problem 1 From PS5

Independent binding sites. A common method of removal of viral and bacterial pathogens is through opsonization. Antibodies can bind viral/bacterial surfaces noncovalently. If enough antibodies bind, the antibody/pathogen complex is recognized by macrophages and engulfed, destroying the pathogen. Let's consider a simplified case study to understand the effective concentrations of antibodies needed for pathogen neutralization. This is a multi-part problem.

Consider human adenovirus 41 Science – this virus has 12 vertexes where antibodies can bind independently. Binding at 8 of these vertexes is sufficient for opsonization by macrophages – e.g. a virus bound by 7 antibodies will not be opsonized, but one bound by 8 to 12 antibodies will be opsonized. Find the specific antibody concentration (in units of μ g/mL) needed to remove 99.5% of the viruses through opsonization. Assume that the concentration of the virus is much lower than the concentration of antibodies, and that the effective dissociation constant for binding a vertex in isolation is given as $K_d^{\mu} = 10nM$. Make the limiting assumption that only one arm of the antibody can bind at a time (monovalent binding). Finally, assume that binding equilibrium is reached rapidly relative to timescales of viral cell entry and macrophage opsonization (this is probably not a great assumption but one we'll make for simplicity).

Problem 1a

Antibody titers (concentrations) are often reported in units of μ g/mL. Convert 100 nM of antibody to units of μ g/mL. Assume that antibodies have an average molecular weight of 150 kDa.

$100nM = 1 \times 10^{-7} mol/L$

$150kDa \cdot 1000g/mol = 150000g/mol$

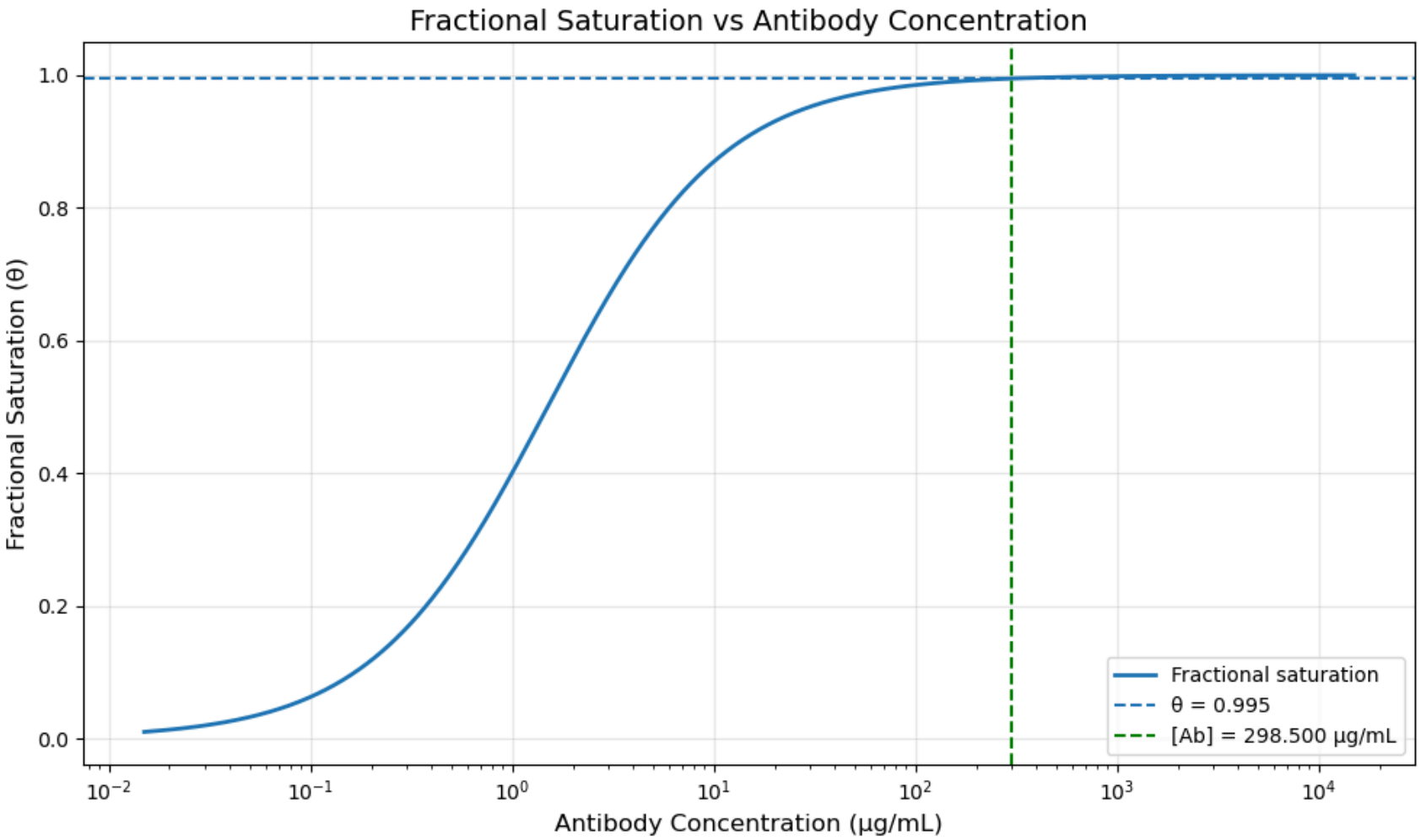
$1 \times 10^{-7} mol/L \times 150000g/mol = 1.5x10^{-2} g/L = 15\mu g/mL$

Problem 1b.

Plot the fractional saturation as a function of antibody concentration in units of μ g/mL. Identify the concentration where the fractional saturation is 0.995. This portion is independent of parts c & d.

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


Kd_nM = 10                  # Dissociation constant (K_d^\mu nM)
MW_antibody = 150000        # Antibody molecular weight (g/mol) ~150 kDa

# Convert Kd to µg/mL
# nM -> mol/L (1e-9)
# mol/L -> g/L (x MW)
# g/L -> µg/mL (x 1000)
Kd_ugml = Kd_nM * MW_antibody * 1e-6

# Antibody concentration range
Ab_conc_nM = np.logspace(-1, 5, 1000)   # 0.1 to 10^5 nM

# Convert to µg/mL
Ab_conc_ugml = Ab_conc_nM * MW_antibody * 1e-6

# Fractional saturation (independent binding)
theta = Ab_conc_nM / (Kd_nM + Ab_conc_nM)

# Plot
plt.figure(figsize=(10,6))

plt.semilogx(
    Ab_conc_ugml,
    theta,
    linewidth=2,
    label='Fractional saturation'
)

# Horizontal line at theta = 0.995
```

```
plt.axhline(
    y=0.995,
    linestyle='--',
    label='θ = 0.995'
)

# Calculate concentration for target saturation
theta_target = 0.995

Ab_nM_at_target = theta_target * Kd_nM / (1 - theta_target)
Ab_ugml_at_target = Ab_nM_at_target * MW_antibody * 1e-6

# Vertical line
plt.axvline(
    x=Ab_ugml_at_target,
    linestyle='--',
    label=f'[Ab] = {Ab_ugml_at_target:.3f} μg/mL',
    color='green'
)

# Labels and formatting
plt.xlabel('Antibody Concentration (μg/mL)', fontsize=12)
plt.ylabel('Fractional Saturation (θ)', fontsize=12)
plt.title('Fractional Saturation vs Antibody Concentration', fontsize=14)

plt.grid(True, alpha=0.3)
plt.legend()
plt.tight_layout()
plt.show()

# Output
print(f"Kd = {Kd_nM} nM = {Kd_ugml:.3f} μg/mL")

print("\nConcentration for θ = 0.995:")
print(f"  [Ab] = {Ab_nM_at_target:.2f} nM")
print(f"  [Ab] = {Ab_ugml_at_target:.3f} μg/mL")
```




```
Kd = 10 nM = 1.500 µg/mL

Concentration for θ = 0.995:
  [Ab] = 1990.00 nM
  [Ab] = 298.500 µg/mL
```

Problem 1c.

Write a single algebraic expression relating the % removal of viruses to the antibody concentration. This should be a single equation with % removal of viruses on the left-hand side and antibody concentration on the right-hand side.

$$P(i) = \binom{N}{i}\theta^i(1-\theta)^{N-i}$$

$$\theta = \frac{[Ab]}{K_d+[Ab]}$$

$$f_{opsonized} = \sum_{i=8}^{1} 2\binom{12}{i}\theta^i(1-\theta)^{12-i}$$

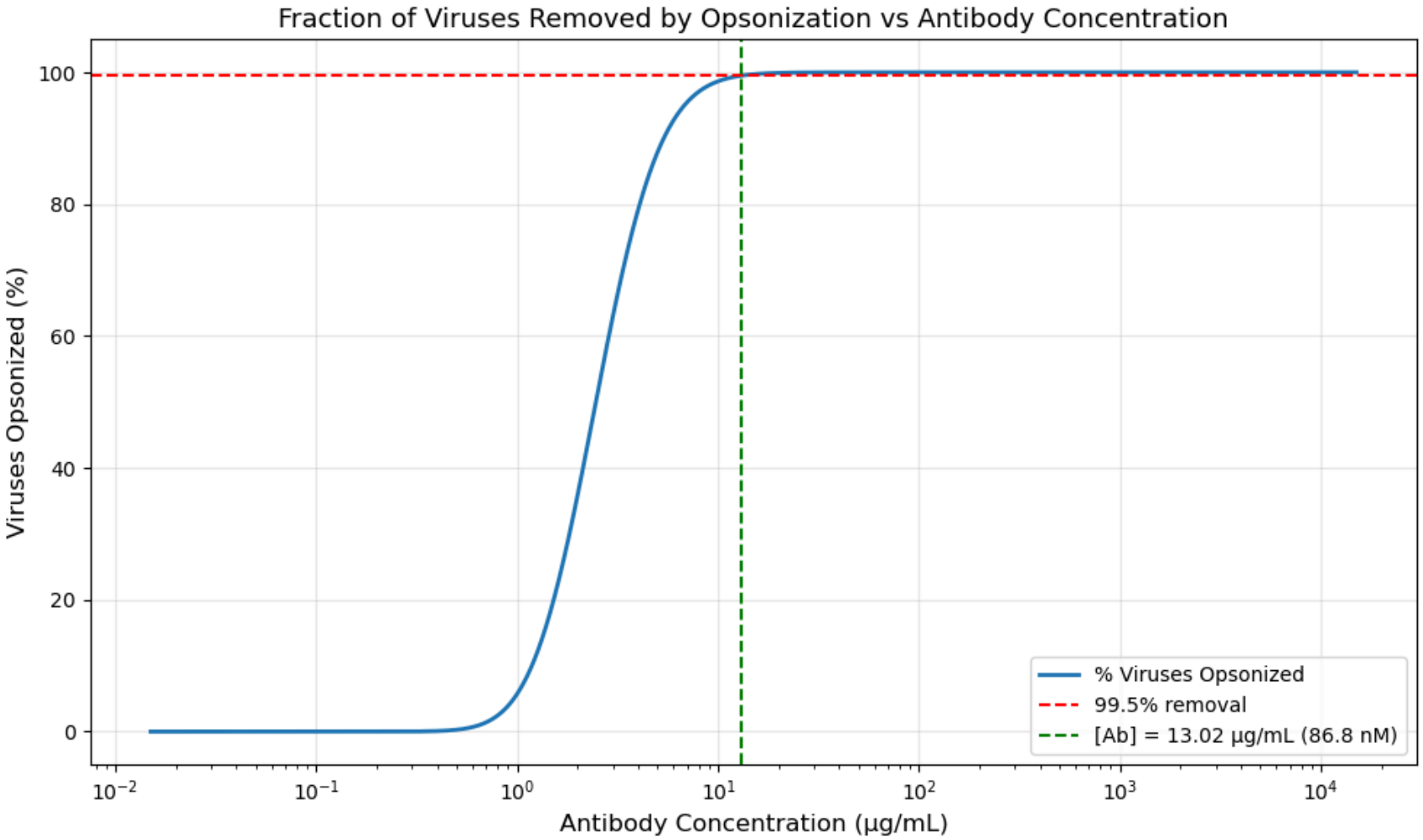
$$\%removed = 100 \cdot \sum_{i=8}^{1} 2\binom{12}{i}\theta^i(1-\theta)^{12-i}$$

Problem 1d

Identify the concentration of antibodies needed for removal 99.5% of the viruses through opsonization. Compare with the value reported in part b.

In [ ]:
```
import numpy as np
import matplotlib.pyplot as plt
from scipy.special import comb

# Constants
N = 12                                  # Number of binding vertices on adenovirus 41
threshold = 8                           # Minimum antibodies needed for opsonization
Kd_nM = 10                              # Dissociation constant (nM)
MW_antibody = 150000                    # ~150 kDa (g/mol)
Kd_ugml = Kd_nM * MW_antibody * 1e-6  # Convert to µg/mL

# Antibody concentration range
Ab_conc_nM   = np.logspace(-1, 5, 10000)
Ab_conc_ugml = Ab_conc_nM * MW_antibody * 1e-6

# Fractional saturation (single site)
theta = Ab_conc_nM / (Kd_nM + Ab_conc_nM)

# Calculate fraction of viruses opsonized (at least 8 antibodies bound)
i_values = np.arange(threshold, N + 1)  # i = 8, 9, 10, 11, 12

f_opsonized = np.zeros_like(theta)
for i in i_values:
```

```
    f_opsonized += comb(N, i, exact=True) * (theta**i) * ((1 - theta)**(N - i))

# Find concentration at 99.5% removal
target = 0.995
idx = np.argmin(np.abs(f_opsonized - target))
Ab_nM_at_target   = Ab_conc_nM[idx]
Ab_ugml_at_target = Ab_conc_ugml[idx]

# Plot
fig, ax = plt.subplots(figsize=(10, 6))

ax.semilogx(Ab_conc_ugml, f_opsonized * 100, linewidth=2, label='% Viruses Opsonized')

ax.axhline(y=99.5, color='red',   linestyle='--', label='99.5% removal')
ax.axvline(x=Ab_ugml_at_target, color='green', linestyle='--',
           label=f'[Ab] = {Ab_ugml_at_target:.2f} µg/mL ({Ab_nM_at_target:.1f} nM)')

ax.set_xlabel('Antibody Concentration (µg/mL)', fontsize=12)
ax.set_ylabel('Viruses Opsonized (%)', fontsize=12)
ax.set_title('Fraction of Viruses Removed by Opsonization vs Antibody Concentration', fontsize=13)
ax.grid(True, alpha=0.3)
ax.legend()
plt.tight_layout()
plt.show()

# Output
print(f"Kd = {Kd_nM} nM = {Kd_ugml:.3f} µg/mL")
print(f"\nPart d — 99.5% virus opsonization:")
print(f"  [Ab] = {Ab_nM_at_target:.2f} nM")
print(f"  [Ab] = {Ab_ugml_at_target:.3f} µg/mL")
print(f"\nPart b — 99.5% site saturation:")
print(f"  [Ab] = {0.995 * Kd_nM / 0.005:.2f} nM")
print(f"  [Ab] = {0.995 * Kd_ugml / 0.005:.3f} µg/mL")
```




```
Kd = 10 nM = 1.500 µg/mL

Part d — 99.5% virus opsonization:
  [Ab] = 86.79 nM
  [Ab] = 13.019 µg/mL

Part b — 99.5% site saturation:
  [Ab] = 1990.00 nM
  [Ab] = 298.500 µg/mL
```

Problem 1 From PS6

Dealing with ligand depletion effects. Wittrup problem 3-27.

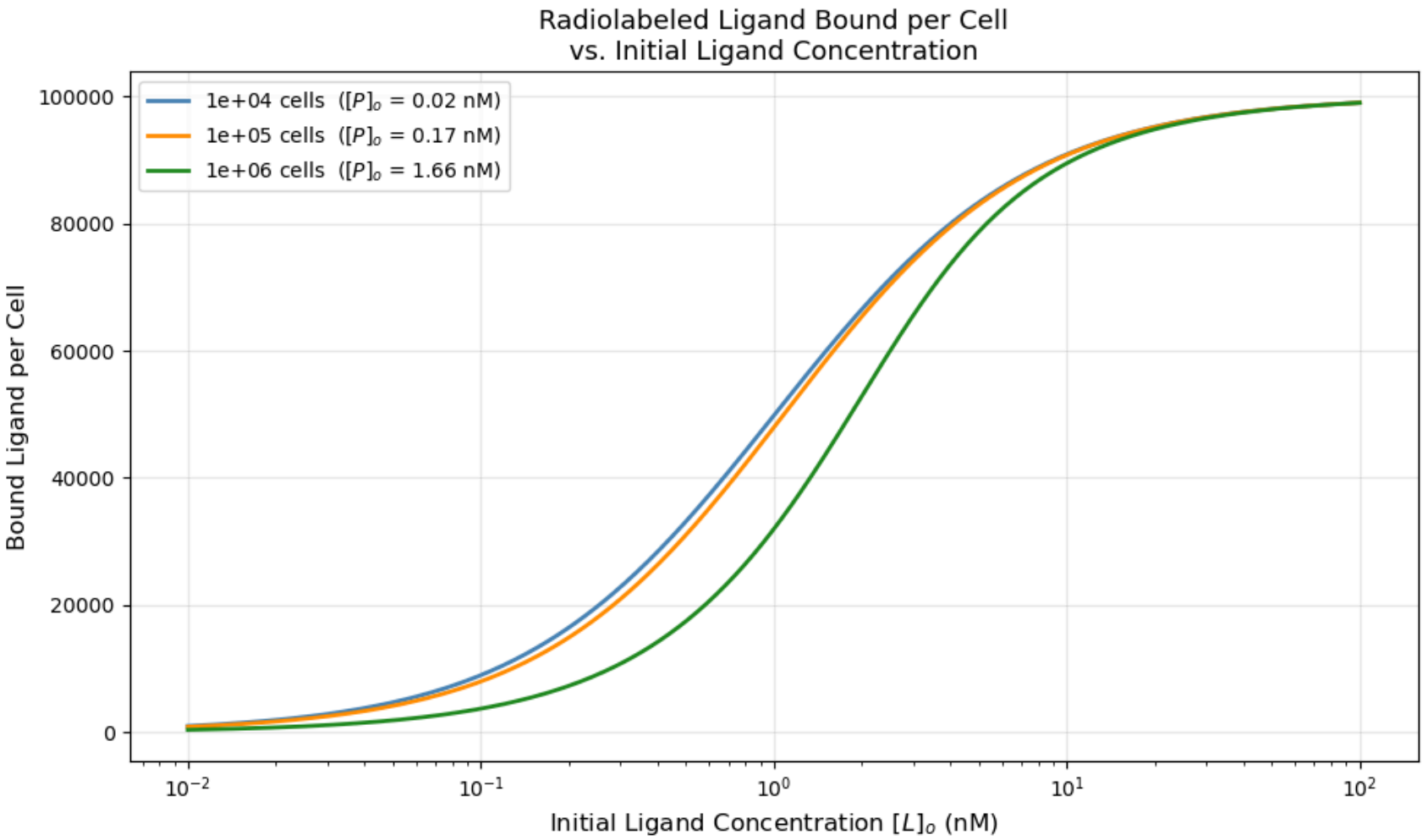
Consider a cell line with 10^5 particular receptors per cell. You have radiolabeled a monovalent ligand, which has an affinity $K_d = 1nM$ for the receptor. Radiolabeled ligand is incubated with 10^4, 10^5, or 10^6 cells in a 100 microliter microplate well, and unbound ligand is washed away.

Problem 1a.

Plot the number of radiolabeled ligand molecules bound per cell as a function of initial ligand concentration (across the range 0.01–100 nM ligand) for each of the cell densities.

In [9]:
```
import numpy as np
import matplotlib.pyplot as plt
# from scipy.optimize import curve_fit

# Constants
Kd   = 1.0                      # nM, dissociation constant
RT        = 1e5                 # receptors per cell
V_L       = 100e-6              # 100 µL
NA        = 6.022e23            # Avogadro's number
cell_densities = [1e4, 1e5, 1e6] # cells per well

# Ligand concentration range (nM)
L0 = np.logspace(-2, 2, 500)   # 0.01 to 100 nM

def bound_per_cell(L0, P0_nM, Kd, RT):
    """
    Rigorous quadratic solution accounting for ligand depletion.
    L0, P0_nM, Kd all in nM.
    Returns bound ligand molecules per cell.
    """
    b = Kd + L0 + P0_nM
    y_eq = (b - np.sqrt(b**2 - 4 * P0_nM * L0)) / (2 * P0_nM)
    return RT * y_eq

fig, ax = plt.subplots(figsize=(10, 6))
```

```
colors = ['steelblue', 'darkorange', 'forestgreen']
bound_data = {}

for n_cells, color in zip(cell_densities, colors):
    # Total receptor concentration in well (nM)
    P0_nM = (n_cells * RT) / (NA * V_L) * 1e9

    B = bound_per_cell(L0, P0_nM, Kd, RT)

    ax.semilogx(L0, B, color=color, linewidth=2,
                label=f'{n_cells:.0e} cells  ($[P]_o$ = {P0_nM:.2f} nM)')

ax.set_xlabel('Initial Ligand Concentration $[L]_o$ (nM)', fontsize=12)
ax.set_ylabel('Bound Ligand per Cell', fontsize=12)
ax.set_title('Radiolabeled Ligand Bound per Cell\nvs. Initial Ligand Concentration', fontsize=13)
ax.grid(True, alpha=0.3)
ax.legend(fontsize=10)
plt.tight_layout()
plt.show()

# Print receptor concentrations for context
print("Receptor concentrations in well:")
for n_cells in cell_densities:
    R_nM = (n_cells * RT) / (NA * V_L) * 1e9
    print(f"  {n_cells:.0e} cells: R_tot = {R_nM:.3f} nM  (vs Kd = {Kd} nM)")
```




```
Receptor concentrations in well:
  1e+04 cells: R_tot = 0.017 nM  (vs Kd = 1.0 nM)
  1e+05 cells: R_tot = 0.166 nM  (vs Kd = 1.0 nM)
  1e+06 cells: R_tot = 1.661 nM  (vs Kd = 1.0 nM)
```

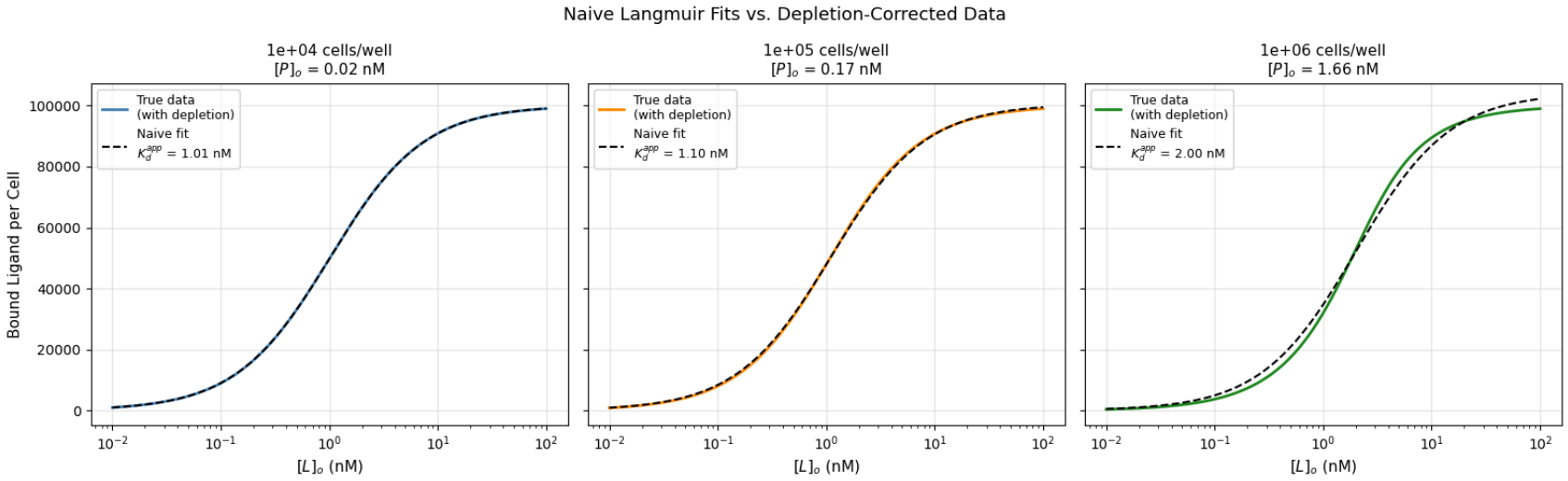
Problem 1b.

Curve-fit a simple K_d (without accounting for depletion effects) to each of the three data sets. What is the apparent binding constant at each of the three cell densities?

In [15]:
```
from scipy.optimize import least_squares

# Naive Langmuir
def langmuir_naive(L0, Bmax, Kd_app):
    return Bmax * L0 / (Kd_app + L0)

def residuals(params, L0, B_data):
    Bmax, Kd_app = params
    return langmuir_naive(L0, Bmax, Kd_app) - B_data

# Plot
fig, axes = plt.subplots(1, 3, figsize=(16, 5), sharey=True)

print(f"{'Density':<15} {'[P]o (nM)':<12} {'Kd_app (nM)':<15}")
for ax, n_cells, color in zip(axes, cell_densities, colors):

    # --- Generate "true" data from depletion-corrected model ---
    P0_nM = (n_cells * RT) / (NA * V_L) * 1e9
    B_true = bound_per_cell(L0, P0_nM, Kd, RT)

    # --- Naive curve fit ---
    init_guess = [RT, 1.0]   # [Bmax, Kd_app]
    res = least_squares(residuals, init_guess,
                        args=(L0, B_true),
                        bounds=([0, 0], [np.inf, np.inf]))
    Bmax_app, Kd_app = res.x

    # --- Plot ---
    ax.semilogx(L0, B_true, color=color, linewidth=2, label='True data\n(with depletion)')
    ax.semilogx(L0, langmuir_naive(L0, Bmax_app, Kd_app),
                'k--', linewidth=1.5,
                label=f'Naive fit\n$K_d^{{app}}$ = {Kd_app:.2f} nM')

    ax.set_xlabel('$[L]_o$ (nM)', fontsize=11)
    ax.set_title(f'{n_cells:.0e} cells/well\n$[P]_o$ = {P0_nM:.2f} nM', fontsize=11)
    ax.grid(True, alpha=0.3)
    ax.legend(fontsize=9)

    print(f"{n_cells:<15.0e} {P0_nM:<12.3f} {Kd_app:<15.3f}")

axes[0].set_ylabel('Bound Ligand per Cell', fontsize=11)
```

```
fig.suptitle('Naive Langmuir Fits vs. Depletion-Corrected Data', fontsize=13)
plt.tight_layout()
plt.show()
```

```
Density         [P]o (nM)    Kd_app (nM)
1e+04           0.017        1.010
1e+05           0.166        1.100
1e+06           1.661        2.002
```



Problem 2

Dealing with insufficient equilibration time. Consider a cell surface titration experiment where soluble, fluorescently conjugated ligand binds to cell surface displayed protein. Autofluorescence (Fmin) was separately at 300 MFI. A different technique was used to measure the monovalent binding dissociation constant at 50 pM and the association rate constant at 1.2e6 $M^{-1}s^{-1}$ at room temperature.

Problem 2A.

Based on this information, determine the amount of time a researcher should incubate the ligand with cells to ensure close-to-equilibrium binding over a labeling concentration of ligand from 10 pM to 50 nM.

In [ ]:
```python
# Parameters
Kd_M    = 50e-12                  # 50 pM in M
kon     = 1.2e6                   # M^-1 s^-1
koff    = Kd_M * kon              # s^-1
t_half_koff = np.log(2) / koff    # intrinsic half-life (no ligand)

# Ligand concentrations to evaluate
L_conc_M = np.logspace(np.log10(10e-12), np.log10(50e-9), 100)

# Time array — go long enough to capture slowest equilibration
t = np.linspace(0, 48 * 3600, 10000)  # 0 to 48 hours in seconds

t_99_results = {}

for L in L_conc_M:
    y_eq  = L / (Kd_M + L)
    k_obs = kon * L + koff
    t_99  = -np.log(0.01) / k_obs   # seconds to reach 99% of y_eq
    label = f"{L*1e12:.1f} pM"

    # if L can be converted to nM for better labeling, do it
    if L >= 1e-9:
        label = f"{L*1e9:.1f} nM"

    t_99_results[label] = (t_99 / 3600, y_eq, k_obs)

L_range_M  = np.logspace(-12, -7, 500)   # 1 pM to 100 nM
k_obs_range = kon * L_range_M + koff
t_99_range  = -np.log(0.01) / k_obs_range / 3600   # hours

# Summary
print(f"\n{'[Concentration]':<12} {'y_eq':<10} {'k_obs (s^-1)':<16} {'time (hours)'}")
print("-" * 52)
for j, (label, (t99_h, y_eq, k_obs)) in enumerate(t_99_results.items()):
    if j % 10 == 0:
        print(f"{label:<12} {y_eq:<10.3f} {k_obs:<16.3e} {t99_h:.2f}")

print(f"\nWorst case is at [L] = 10 pM: incubate for {t_99_results['10.0 pM'][0]:.2f} hours")
```

```
[Concentration] y_eq       k_obs (s^-1)     time (hours)
-----------------------------------------------------
10.0 pM      0.167      7.200e-05        17.77
23.6 pM      0.321      8.837e-05        14.48
55.9 pM      0.528      1.271e-04        10.07
132.1 pM     0.725      2.185e-04        5.85
312.3 pM     0.862      4.347e-04        2.94
738.2 pM     0.937      9.458e-04        1.35
1.7 nM       0.972      2.154e-03        0.59
4.1 nM       0.988      5.010e-03        0.26
9.8 nM       0.995      1.176e-02        0.11
23.1 nM      0.998      2.772e-02        0.05

Worst case is at [L] = 10 pM: incubate for 17.77 hours
```

Problem 2B.

The researcher did not do this calculation and instead labeled ligand with cells for only 10 minutes at room temperature. The researcher measured a fluorescence of cells of 30,500 MFI at the highest labeling concentration of 50 nM. Based on these facts, determine the likely measured fluorescence of cells for the following labeling concentrations: 10 pM, 20 pM, 40 pM, 80 pM, 160 pM, 320 pM, 1 nM, 2 nM, 4 nM, 8 nM, 16 nM.

In [ ]:
```python
# Parameters
Kd_M = 50e-12       # 50 pM
kon = 1.2e6         # M^-1 s^-1
koff = Kd_M * kon   # s^-1
Fmin = 300          # MFI
t = 10 * 60         # 10 minutes in seconds
L_anchor = 50e-9
F_anchor = 30500

y_eq_anchor  = L_anchor / (Kd_M + L_anchor)
k_obs_anchor = kon * L_anchor + koff
y_t_anchor   = y_eq_anchor * (1 - np.exp(-k_obs_anchor * t))

# F = Fmin + (Fmax - Fmin) * y(t)  -->  solve for Fmax
Fmax = Fmin + (F_anchor - Fmin) / y_t_anchor

# Labeling concentrations
```

```
L_conc_M = np.array([10e-12, 20e-12, 40e-12, 80e-12, 160e-12,
                     320e-12, 1e-9, 2e-9, 4e-9, 8e-9, 16e-9])
L_labels = ['10 pM','20 pM','40 pM','80 pM','160 pM',
            '320 pM','1 nM','2 nM','4 nM','8 nM','16 nM']

# Find the point where the flourescence is closest to the anchor point (30500 MFI)
min_diff = float('inf')
closest_label = None
closest_F = None
F_values = []
for L, label in zip(L_conc_M, L_labels):
    y_eq = L / (Kd_M + L)
    k_obs = kon * L + koff
    y_t = y_eq * (1 - np.exp(-k_obs * t))
    F = Fmin + (Fmax - Fmin) * y_t
    F_values.append(F)

    diff = abs(F - F_anchor)
    if diff < min_diff:
        min_diff = diff
        closest_label = label
        closest_F = F

print(f"Closest fluorescence to anchor point: {closest_label} with F={closest_F:.1f} MFI")
```

```
Closest fluorescence to anchor point: 16 nM with F=30435.7 MFI
```

Problem 2C.

Using the fluorescent values in part b, use curve fitting to calculate an estimated Kd for this experiment. What is the percentage error for this calculation compared with the ‘true’ value of 50 pM?

In [30]:
```
from scipy.optimize import curve_fit

# equilibrium model
def equilibrium_binding(L, Fmax, Kd):
    return Fmin + (Fmax - Fmin) * (L / (Kd + L))

# curve fit
popt, pcov = curve_fit(
```

```
    equilibrium_binding,
    L_conc_M,
    F_values,
    p0=[Fmax, 100e-12]
)

Fmax_fit, Kd_fit = popt

# percent error
Kd_true = 50e-12 # 50 pM in M
percent_error = abs(Kd_fit - Kd_true) / Kd_true * 100

print(f"Estimated Kd = {Kd_fit*1e12:.2f} pM, or {Kd_fit*1e9:.2f} nM")
print(f"Percent error = {percent_error:.2f}%")
```

```
Estimated Kd = 1158.95 pM, or 1.16 nM
Percent error = 2217.90%
```

Problem 3

Cooperativity between binding sites. Fitting models to data. In Biochemistry intro courses, you are often taught about positive cooperativity between binding sites. Oxygen binding by hemoglobin is the canonical example. Less commonly discussed is negative cooperativity between binding sites. This negative cooperativity can often be modeled by the Hill equation, where the Hill coefficient n is modeled as less than one. Consider the following experimentally determined dataset, using cell surface titrations of a fluorescently conjugated monomeric growth factor with a monomeric receptor. The fluorescence measurements were taken in experimental triplicate (see appendix).

Raw data used for problem #3.

Mean fluorescence intensity [RFU]

Ligand [nM]	Replicate 1	Replicate 2	Replicate 3
0.2	4336	4138	4046
0.5	6138	6087	6045

Ligand [nM]	Replicate 1	Replicate 2	Replicate 3
1.1	7960	7990	8494
2.5	11299	10653	11157
5.7	14256	14134	15092
13.2	18322	19527	18284
30.4	23534	23374	22476
70.2	29414	27520	27852
162.2	30477	33198	32333
374.8	34975	34866	37999
865.8	36478	37096	35836
2000.0	40237	40956	38818

Problem 3A.

Using a model of monovalent binding, determine the best fit Kd by minimizing chi squared metric. Explicitly state how you are dealing with measurement uncertainty.

In [45]:
```
from scipy.stats import chi2 as chi2_dist

# Data
ligand_conc = np.array([0.2, 0.5, 1.1, 2.5, 5.7, 13.2,
                        30.4, 70.2, 162.2, 374.8, 865.8, 2000.0])

run1 = np.array([4336, 6138, 7960, 11299, 14256, 18322,
                 23534, 29414, 30477, 34975, 36478, 40237])
run2 = np.array([4138, 6087, 7990, 10653, 14134, 19527,
                 23374, 27520, 33198, 34866, 37096, 40956])
run3 = np.array([4046, 6045, 8494, 11157, 15092, 18284,
                 22476, 27852, 32333, 37999, 35836, 38818])

all_runs = np.vstack((run1, run2, run3))
y_means  = np.mean(all_runs, axis=0)
```

```
y_stds   = np.std(all_runs,  axis=0, ddof=1)   # sample std — our sigma_i

# Flatten all 36 points for fitting
x_flat     = np.tile(ligand_conc, 3)
y_flat     = all_runs.flatten()
sigma_flat = np.tile(y_stds, 3)

# Model: monovalent binding
def binding_model(x, Fmin, Fmax, Kd):
    return Fmin + (Fmax - Fmin) * x / (x + Kd)

# Multiple initial guesses to avoid local minima
init_guesses = [
    [3000, 42000, 50],
    [2000, 45000, 100],
    [4000, 40000, 20],
]

best_res  = None
best_chi2 = np.inf

for p0 in init_guesses:
    try:
        popt, pcov = curve_fit(binding_model, x_flat, y_flat,
                               p0=p0, sigma=sigma_flat,
                               absolute_sigma=True,
                               maxfev=10000)
        residuals        = y_flat - binding_model(x_flat, *popt)
        chi2_val         = np.sum((residuals / sigma_flat)**2)
        if chi2_val < best_chi2:
            best_chi2 = chi2_val
            best_res  = (popt, pcov)
    except Exception:
        continue

popt, pcov = best_res
Fmin_fit, Fmax_fit, Kd_fit = popt

# Statistics
n_data = len(y_flat)      # 36
n_params = 3                # Fmin, Fmax, Kd
dof = n_data - n_params
```

```
residuals = y_flat - binding_model(x_flat, *popt)
chi2_val = np.sum((residuals / sigma_flat)**2)
reduced_chi2 = chi2_val / dof
p_value = chi2_dist.sf(chi2_val, dof)

popt_mono = popt
reduced_chi2_mono = reduced_chi2
p_value_mono = p_value
dof_mono = dof

# Output
print(f"  Monovalent Binding Fit Results")
print(f"  Fmin:             {Fmin_fit:.1f} RFU")
print(f"  Fmax:             {Fmax_fit:.1f} RFU")
print(f"  Kd:               {Kd_fit:.2f} nM")
print(f"  Chi-squared:      {chi2_val:.2f}")
print(f"  DOF (n-m):        {dof}  ({n_data} pts - {n_params} params)")
print(f"  Reduced chi-sq:   {reduced_chi2:.3f}")
print(f"  P-value:          {p_value:.4e}")

if p_value < 0.05:
    print("  CONCLUSION: Reject model (p < 0.05)")
else:
    print("  CONCLUSION: Fail to reject model (p > 0.05)")

print(f'\nMeasurement uncertainty is handled using the sample standard deviation'
      f' across triplicates as sigma_i in the chi-squared minimization.')
```

```
 Monovalent Binding Fit Results
 Fmin:             4895.5 RFU
 Fmax:             35688.0 RFU
 Kd:               13.37 nM
 Chi-squared:      486.85
 DOF (n-m):        33  (36 pts - 3 params)
 Reduced chi-sq:   14.753
 P-value:          3.8327e-82
 CONCLUSION: Reject model (p < 0.05)

Measurement uncertainty is handled using the sample standard deviation across triplicates as sigma_i in the
chi-squared minimization.
```

Problem 2B.

Repeat step a but instead use a model of cooperative binding. Determine the best fit EC50 and Hill coefficient n.

In [46]:
```
# Model: Hill / cooperative binding
def hill_model(x, Fmin, Fmax, EC50, n):
    return Fmin + (Fmax - Fmin) * (x**n / (x**n + EC50**n))

# Multiple initial guesses
init_guesses = [
    [3000, 42000, 50, 1.0],
    [2000, 45000, 100, 0.5],
    [4000, 40000, 20, 0.7],
]

best_res  = None
best_chi2 = np.inf

for p0 in init_guesses:
    try:
        popt, pcov = curve_fit(hill_model, x_flat, y_flat,
                               p0=p0, sigma=sigma_flat,
                               absolute_sigma=True,
                               bounds=([0, 0, 0, 0], [np.inf, np.inf, np.inf, np.inf]),
                               maxfev=10000)
        residuals = y_flat - hill_model(x_flat, *popt)
        chi2_val  = np.sum((residuals / sigma_flat)**2)
        if chi2_val < best_chi2:
            best_chi2 = chi2_val
            best_res  = (popt, pcov)
    except Exception:
        continue

popt, pcov = best_res
Fmin_fit, Fmax_fit, EC50_fit, n_fit = popt

# Statistics
n_data   = len(y_flat)      # 36
n_params = 4                # Fmin, Fmax, EC50, n
```

```
dof      = n_data - n_params

residuals    = y_flat - hill_model(x_flat, *popt)
chi2_val     = np.sum((residuals / sigma_flat)**2)
reduced_chi2 = chi2_val / dof
p_value      = chi2_dist.sf(chi2_val, dof)

popt_hill= popt
reduced_chi2_hill = reduced_chi2
p_value_hill = p_value
dof_hill = dof

# Output
print(f"  Hill Model Fit Results")
print(f"  Fmin:             {Fmin_fit:.1f} RFU")
print(f"  Fmax:             {Fmax_fit:.1f} RFU")
print(f"  EC50:             {EC50_fit:.2f} nM")
print(f"  Hill coeff n:     {n_fit:.3f}")
print(f"  Chi-squared:      {chi2_val:.2f}")
print(f"  DOF (n-m):        {dof}  ({n_data} pts - {n_params} params)")
print(f"  Reduced chi-sq:   {reduced_chi2:.3f}")
print(f"  P-value:          {p_value:.4e}")

if p_value < 0.05:
    print("  CONCLUSION: Reject model (p < 0.05)")
else:
    print("  CONCLUSION: Fail to reject model (p > 0.05)")

print(f'\nMeasurement uncertainty is handled using the sample standard deviation'
      f' across triplicates as sigma_i in the chi-squared minimization.')
```

```
Hill Model Fit Results
Fmin:             501.0 RFU
Fmax:             43501.9 RFU
EC50:             24.30 nM
Hill coeff n:     0.490
Chi-squared:      35.54
DOF (n-m):        32  (36 pts - 4 params)
Reduced chi-sq:   1.110
P-value:          3.0525e-01
CONCLUSION: Fail to reject model (p > 0.05)
```

Measurement uncertainty is handled using the sample standard deviation across triplicates as sigma_i in the chi-squared minimization.

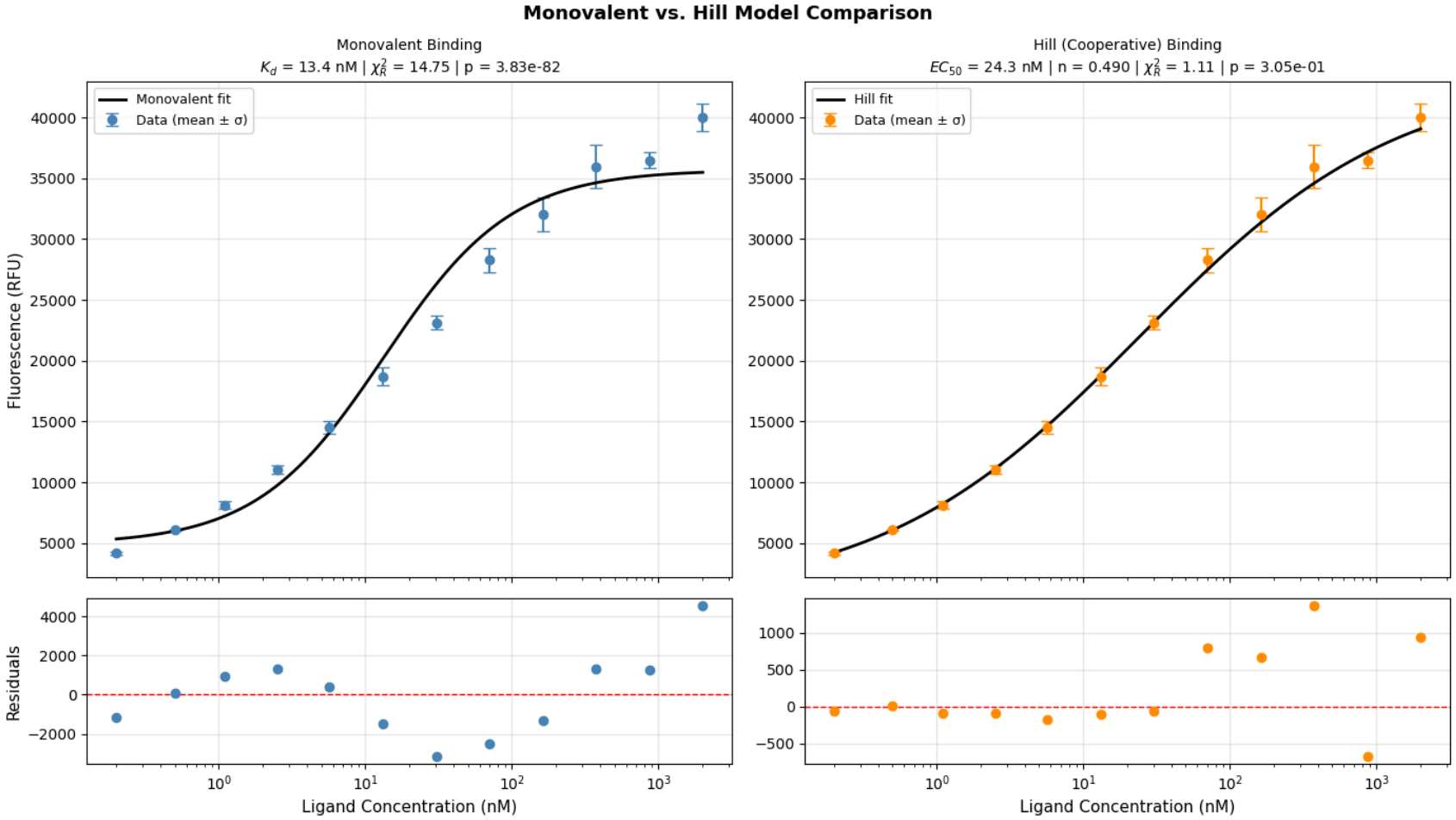
Problem 2C.

Show qualitatively (by analysis of residuals) and quantitatively (by a reduced chi squared test) that the cooperative model is more likely to be correct.

In [52]:
```
import matplotlib.pyplot as plt

# Smooth curves
x_smooth      = np.logspace(np.log10(min(ligand_conc)), np.log10(max(ligand_conc)), 500)
y_smooth_mono = binding_model(x_smooth, *popt_mono)
y_smooth_hill = hill_model(x_smooth, *popt_hill)

# Residuals at each concentration (means)
res_mono = y_means - binding_model(ligand_conc, *popt_mono)
res_hill = y_means - hill_model(ligand_conc, *popt_hill)

# Plot
fig, axes = plt.subplots(2, 2, figsize=(14, 8),
                         gridspec_kw={'height_ratios': [3, 1]},
                         sharex='col')

# Monovalent fit
axes[0, 0].errorbar(ligand_conc, y_means, yerr=y_stds,
                    fmt='o', color='steelblue', capsize=4, label='Data (mean ± σ)')
axes[0, 0].plot(x_smooth, y_smooth_mono, 'k-', linewidth=2, label='Monovalent fit')
axes[0, 0].set_xscale('log')
```

```
axes[0, 0].set_ylabel('Fluorescence (RFU)', fontsize=11)
axes[0, 0].set_title(f'Monovalent Binding\n'
                     f'$K_d$ = {popt_mono[2]:.1f} nM | '
                     f'$\\chi^2_R$ = {reduced_chi2_mono:.2f} | '
                     f'p = {p_value_mono:.2e}', fontsize=10)
axes[0, 0].legend(fontsize=9)
axes[0, 0].grid(True, alpha=0.3)

# Hill fit
axes[0, 1].errorbar(ligand_conc, y_means, yerr=y_stds,
                    fmt='o', color='darkorange', capsize=4, label='Data (mean ± σ)')
axes[0, 1].plot(x_smooth, y_smooth_hill, 'k-', linewidth=2, label='Hill fit')
axes[0, 1].set_xscale('log')
axes[0, 1].set_title(f'Hill (Cooperative) Binding\n'
                     f'$EC_{{50}}$ = {popt_hill[2]:.1f} nM | n = {popt_hill[3]:.3f} | '
                     f'$\\chi^2_R$ = {reduced_chi2_hill:.2f} | '
                     f'p = {p_value_hill:.2e}', fontsize=10)
axes[0, 1].legend(fontsize=9)
axes[0, 1].grid(True, alpha=0.3)

# Monovalent residuals
axes[1, 0].scatter(ligand_conc, res_mono, color='steelblue', zorder=3)
axes[1, 0].axhline(0, color='red', linestyle='--', linewidth=1)
axes[1, 0].set_xscale('log')
axes[1, 0].set_xlabel('Ligand Concentration (nM)', fontsize=11)
axes[1, 0].set_ylabel('Residuals', fontsize=11)
axes[1, 0].grid(True, alpha=0.3)

# Hill residuals
axes[1, 1].scatter(ligand_conc, res_hill, color='darkorange', zorder=3)
axes[1, 1].axhline(0, color='red', linestyle='--', linewidth=1)
axes[1, 1].set_xscale('log')
axes[1, 1].set_xlabel('Ligand Concentration (nM)', fontsize=11)
axes[1, 1].grid(True, alpha=0.3)

plt.suptitle('Monovalent vs. Hill Model Comparison', fontsize=13, fontweight='bold')
plt.tight_layout()
plt.show()


print(f'Monovalent:Kd = {popt_mono[2]:.1f} nM | '
      f'χ²_R = {reduced_chi2_mono:.2f} | p = {p_value_mono:.2e}')
```

```
print(f'\nHill: EC50 = {popt_hill[2]:.1f} nM | '
      f'n = {popt_hill[3]:.3f} | '
      f'χ²_R = {reduced_chi2_hill:.2f} | p = {p_value_hill:.2e}')
print(f'\nHill model is preferred: lower χ²_R and higher p-value.')
print(f'\nn < 1 indicates negative cooperativity.')
print(f'\nHill residuals scatter randomly around zero.')
```



```
Monovalent:Kd = 13.4 nM | χ²_R = 14.75 | p = 3.83e-82

Hill: EC50 = 24.3 nM | n = 0.490 | χ²_R = 1.11 | p = 3.05e-01

Hill model is preferred: lower χ²_R and higher p-value.

n < 1 indicates negative cooperativity.

Hill residuals scatter randomly around zero.
```

Problem 2D.

There are many possible reasons for observed negative cooperativity in binding. Perhaps the most common reason is when non-equivalent binding sites are present, each with a separate K_d. The fractional saturation y can be modeled generally as:

$$y = \sum_{i=1}^{n} \frac{f_i [L]_o}{[L]_o + K_{D,i}} ; \text{where} \sum_{i=1}^{n} f_i = 1$$

In this case (monomeric receptor, monomeric growth factor), the presence of non-equivalent binding sites is almost certainly the explanation for the observed experimental results. Find a set of most likely parameter values f_i and $K_{D,i)}$ (you will have to determine the number of binding sites) for which you get equivalent chi squared metric with part a. Then, determine which model (negative cooperative or non-equivalent binding sites) best fits the experimental data.

In [ ]:
```python
def neq_model_nsite(x, Fmin, Fmax, *args):
    """
    General n-site non-equivalent binding model.
    args = [f1, f2, ..., f(n-1), Kd1, Kd2, ..., Kdn]
    fn = 1 - sum(f1...f(n-1)) enforced implicitly.
    """
    n = (len(args) + 1) // 2   # number of sites
    fs = np.array(list(args[:n-1]) + [1 - sum(args[:n-1])])
    Kds = np.array(args[n-1:])
    y = sum(f * x / (x + Kd) for f, Kd in zip(fs, Kds))
    return Fmin + (Fmax - Fmin) * y

# Iterate over number of sites
results = {}
```

```
for n_sites in range(1,15):
    n_free_f = n_sites - 1
    n_params = 2 + n_free_f + n_sites
    dof = len(y_flat) - n_params

    if dof <= 0:
        print(f"  {n_sites} sites: not enough DOF, skipping.")
        continue


    lb = [0, 0] + [0] * n_free_f + [0] * n_sites
    ub = [np.inf, np.inf] + [1] * n_free_f + [np.inf] * n_sites

    # Generate multiple initial guesses
    init_guesses = []
    for _ in range(20):
        fs_guess  = np.random.dirichlet(np.ones(n_sites))[:-1]
        Kds_guess = np.random.choice(ligand_conc, n_sites)
        init_guesses.append([3000, 42000] + list(fs_guess) + list(Kds_guess))

    best_chi2 = np.inf
    best_popt = None

    for p0 in init_guesses:
        try:
            popt, _ = curve_fit(neq_model_nsite, x_flat, y_flat,
                                p0=p0, sigma=sigma_flat,
                                absolute_sigma=True,
                                bounds=(lb, ub),
                                maxfev=20000)
            # Check fs sum constraint is satisfied
            n_free = n_sites - 1
            fs_fit = list(popt[2:2+n_free])
            if sum(fs_fit) > 1:
                continue
            chi2_val = np.sum(((y_flat - neq_model_nsite(x_flat, *popt)) / sigma_flat)**2)
            if chi2_val < best_chi2:
                best_chi2 = chi2_val
                best_popt = popt
        except Exception:
            continue
```

```
    if best_popt is None:
        print(f" {n_sites} sites: fit failed.")
        continue

    reduced_chi2 = best_chi2 / dof
    p_value      = chi2_dist.sf(best_chi2, dof)

    # Reconstruct fs and Kds for display
    fs_fit  = list(best_popt[2:2+n_free_f]) + [1 - sum(best_popt[2:2+n_free_f])]
    Kds_fit = list(best_popt[2+n_free_f:])

    results[n_sites] = {
        'popt': best_popt, 'chi2': best_chi2,
        'reduced_chi2': reduced_chi2, 'p_value': p_value,
        'dof': dof, 'fs': fs_fit, 'Kds': Kds_fit,
        'n_params': n_params
    }

# Summary table
print(f'  Non-Equivalent Sites — Model Selection')
print(f'  {"Sites":<8} {"n_params":<10} {"DOF":<6} {"χ²_R":<10} {"p-value"}')
for n, r in results.items():
    print(f'  {n:<8} {r["n_params"]:<10} {r["dof"]:<6} '
          f'{r["reduced_chi2"]:<10.3f} {r["p_value"]:.4e}')
print(f'  Optimal: fewest sites where χ²_R ≈ 1 and p > 0.05')

# Identify best model: p > 0.05 and closest reduced chi-squared to 1
best_model = None
closest_difference = float('inf')
for n, r in results.items():
    if r['p_value'] > 0.05:
        diff = abs(r['reduced_chi2'] - 1.0)
        if diff < closest_difference:
            closest_difference = diff
            best_model = (n, r)

if best_model is not None:
    n_best, r_best = best_model
    print(f'\nBest model: {n_best} non-equivalent sites')
    print(f'  Fmin: {r_best["popt"][0]:.1f} RFU')
    print(f'  Fmax: {r_best["popt"][1]:.1f} RFU')
    for i in range(n_best):
```

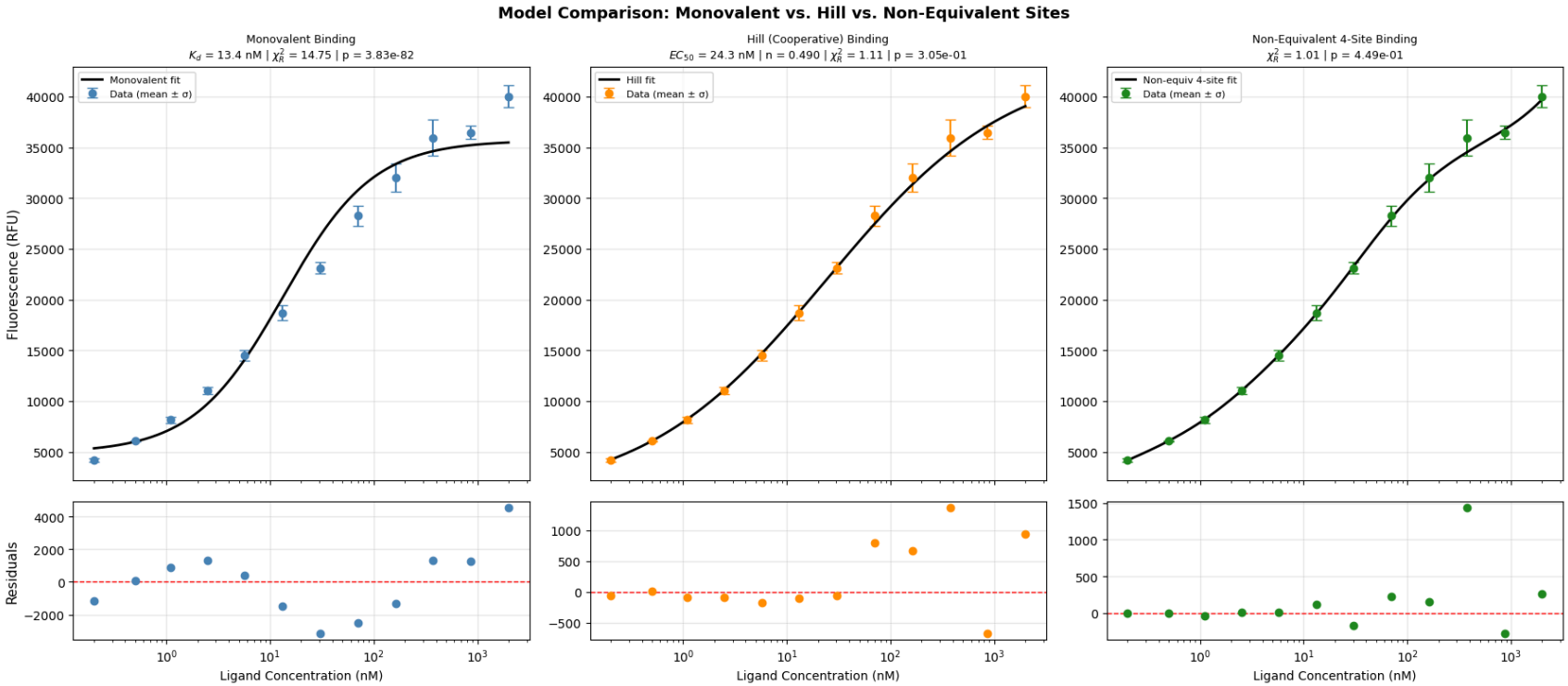
```
        print(f'  Site {i+1}: f={r_best["fs"][i]:.3f}, Kd={r_best["Kds"][i]:.2f} nM')
    print(f'  χ²_R: {r_best["reduced_chi2"]:.3f}')
    print(f'  p-value: {r_best["p_value"]:.4e}')
```

```
Non-Equivalent Sites — Model Selection
Sites    n_params   DOF    χ²_R       p-value
1        3          33     14.753     3.8327e-82
2        5          31     1.516      3.2846e-02
3        7          29     1.033      4.1552e-01
4        9          27     1.010      4.4891e-01
5        11         25     1.092      3.4070e-01
6        13         23     1.228      2.0704e-01
7        15         21     1.319      1.4891e-01
8        17         19     1.569      5.4182e-02
9        19         17     1.750      2.8151e-02
10       21         15     1.985      1.2786e-02
11       23         13     2.255      5.9111e-03
12       25         11     2.596      2.6630e-03
13       27         9      3.363      3.9555e-04
14       29         7      4.301      9.0872e-05
Optimal: fewest sites where χ²_R ≈ 1 and p > 0.05

Best model: 4 non-equivalent sites
  Fmin: 0.0 RFU
  Fmax: 113394.9 RFU
  Site 1: f=0.045, Kd=0.10 nM
  Site 2: f=0.685, Kd=32780.39 nM
  Site 3: f=0.089, Kd=2.64 nM
  Site 4: f=0.181, Kd=40.04 nM
  χ²_R: 1.010
  p-value: 4.4891e-01
```

In [60]:

```
# Best non-equivalent sites model smooth curve & residuals
n_best, r_best = best_model
y_smooth_neq = neq_model_nsite(x_smooth, *r_best['popt'])
res_neq      = y_means - neq_model_nsite(ligand_conc, *r_best['popt'])

# Plot — 3 columns
fig, axes = plt.subplots(2, 3, figsize=(18, 8),
                         gridspec_kw={'height_ratios': [3, 1]},
                         sharex='col')
```

```
plot_configs = [
    (axes[0,0], axes[1,0], y_smooth_mono, res_mono, 'steelblue',   'Monovalent fit',
     f'Monovalent Binding\n$K_d$ = {popt_mono[2]:.1f} nM | '
     f'$\\chi^2_R$ = {reduced_chi2_mono:.2f} | p = {p_value_mono:.2e}'),

    (axes[0,1], axes[1,1], y_smooth_hill, res_hill, 'darkorange',  'Hill fit',
     f'Hill (Cooperative) Binding\n$EC_{{50}}$ = {popt_hill[2]:.1f} nM | '
     f'n = {popt_hill[3]:.3f} | '
     f'$\\chi^2_R$ = {reduced_chi2_hill:.2f} | p = {p_value_hill:.2e}'),

    (axes[0,2], axes[1,2], y_smooth_neq,  res_neq,  'forestgreen', f'Non-equiv {n_best}-site fit',
     f'Non-Equivalent {n_best}-Site Binding\n'
     f'$\\chi^2_R$ = {r_best["reduced_chi2"]:.2f} | p = {r_best["p_value"]:.2e}'),
]

for ax_top, ax_bot, y_smooth, res, color, fit_label, title in plot_configs:
    ax_top.errorbar(ligand_conc, y_means, yerr=y_stds,
                    fmt='o', color=color, capsize=4, label='Data (mean ± σ)')
    ax_top.plot(x_smooth, y_smooth, 'k-', linewidth=2, label=fit_label)
    ax_top.set_xscale('log')
    ax_top.set_title(title, fontsize=9)
    ax_top.legend(fontsize=8)
    ax_top.grid(True, alpha=0.3)

    ax_bot.scatter(ligand_conc, res, color=color, zorder=3)
    ax_bot.axhline(0, color='red', linestyle='--', linewidth=1)
    ax_bot.set_xscale('log')
    ax_bot.set_xlabel('Ligand Concentration (nM)', fontsize=10)
    ax_bot.grid(True, alpha=0.3)

axes[0, 0].set_ylabel('Fluorescence (RFU)', fontsize=11)
axes[1, 0].set_ylabel('Residuals', fontsize=11)

plt.suptitle('Model Comparison: Monovalent vs. Hill vs. Non-Equivalent Sites',
             fontsize=13, fontweight='bold')
plt.tight_layout()
plt.show()
```



Location of Jupyter Notebook

This file was created in jupyter notebook, rendered into `.html` through `jupyter nbconvert --to html PS6/PS6.ipynb` then the `.html` file was saved as a `.pdf`. The location of all said files are github.com/caterer-z-t/CHEN_5150/PS6

Please contact ztcaterer@colorado.edu if there are any issues or concerns.

Thanks :-)