

Additional File: Simulation Source Code (je_sim)

Mechanistic simulator for Culex / Japanese encephalitis with extreme-precipitation flushing-and-rebound

Page 1 of 6

```
"""
```

```
Culex / Japanese encephalitis mechanistic simulator with extreme-precipitation  
flushing-and-rebound (PFI-X) and an urban-vulnerability overlay.
```

```
Weather forcing is a REPRODUCIBLE synthetic climatology calibrated to Guangzhou  
(mean annual 22.4 C, seasonal amplitude 7.1 C, monsoon precipitation regime),  
with discrete Extreme Precipitation Events (EPE) injected. Keeps the experiment  
controlled and fully reproducible; accepts real ERA5 without code changes.  
All reported numbers are produced by this script.
```

```
"""
```

```
import numpy as np  
from numpy.linalg import lstsq  
import json, math  
import matplotlib  
matplotlib.use("Agg")  
import matplotlib.pyplot as plt
```

```
rng = np.random.default_rng(20260817)  
N_GRID = 10  
WEEKS = 416  
DAYS = WEEKS * 7  
DOY = np.arange(DAYS) % 365
```

```
# ----- weather forcing (synthetic, Guangzhou-calibrated) -----
```

```
def build_weather(grid_offset):  
    Tmean = 22.4 + grid_offset[0]  
    Amp = 7.1 + grid_offset[1]  
    base = Tmean + Amp * np.cos(2*np.pi*(np.arange(DAYS)-200)/365.0)  
    noise = np.zeros(DAYS)  
    eps = rng.normal(0, 1.4, DAYS)  
    for d in range(1, DAYS):  
        noise[d] = 0.6*noise[d-1] + eps[d]  
    T = base + noise  
    mon = 0.5 + 0.5*np.cos(2*np.pi*(np.arange(DAYS)-180)/365.0)  
    p_wet = np.clip(0.20 + 0.35*mon, 0.05, 0.7)  
    wet = rng.random(DAYS) < p_wet  
    amount = np.zeros(DAYS)  
    amount[wet] = rng.gamma(0.8, 7.0, size=wet.sum())  
    epe_days = np.zeros(DAYS, dtype=bool)  
    for yr in range(8):  
        for _ in range(3):  
            start = yr*365 + rng.integers(120, 240)  
            for k in range(3):  
                if start+k < DAYS:  
                    epe_days[start+k] = True  
                    amount[start+k] = 90.0 + 45.0*rng.random()  
    return T, amount, epe_days
```

```
# ----- Culex PFI-X abundance -----
```

```
def briere(T, Tmin=9.0, Tmax=40.0, a=1.13e-4):  
    T = np.asarray(T, float)  
    r = np.zeros_like(T)
```

Additional File: Simulation Source Code (je_sim)

Mechanistic simulator for Culex / Japanese encephalitis with extreme-precipitation flushing-and-rebound

Page 2 of 6

```
m = (T > Tmin) & (T < Tmax)
r[m] = a * T[m] * (T[m]-Tmin) * np.sqrt(Tmax - T[m])
return np.clip(r / np.max(r), 0, 1)

def gaussian_surv(T, Tc=27.0, sigma=7.0, smax=0.95):
    return smax * np.exp(-(T-Tc)**2)/(2*sigma**2))

def culex_index(Tdaily, Rdaily, epe_days):
    Tw = Tdaily.reshape(WEEKS, 7).mean(axis=1)
    Rw = Rdaily.reshape(WEEKS, 7).sum(axis=1)
    epe_w = epe_days.reshape(WEEKS, 7).any(axis=1)
    rT = briere(Tw); surv = gaussian_surv(Tw)
    fR = 1.0 + 2.5*(1.0 - np.exp(-(np.maximum(Rw-5,0))/15.0)) # baseline monotone
    habitat = 0.6 + 0.4*(Rw/np.percentile(Rw, 90)).clip(0, 1.2)
    flush = 1.0 - 0.6*epe_w.astype(float) # immediate dip
    rebound = np.ones(WEEKS)
    for t in range(WEEKS):
        for s in range(max(0,t-3), t):
            if epe_w[s]:
                rebound[t] += 0.5*np.exp(-(t-s)/1.4) # 1-3 wk rebound
    C = rT * surv * fR * habitat * flush * rebound
    Craw = C.copy()
    C = C / C.mean()
    return Tw, Rw, epe_w, C, Crawl

# ----- JE three-host SEIR (weekly, substepped) -----
def run_seir(C, epe_w, vuln, sub=7):
    H = 100000.0
    P = 1500.0 * (0.5 + vuln)
    em0 = 600.0
    muM = 0.30
    psi = 0.25 # host -> mosquito transmission scalar
    phi_h = 0.10 # mosquito -> human
    phi_p = 0.22 # mosquito -> pig
    vacc = np.clip(0.65 - 0.45*vuln, 0.05, 0.95)
    import_p = 0.004 * P
    dt = 1.0/sub
    M = np.array([em0*C[0]/muM, 0.0, 0.0])
    Pc = np.array([P*0.98, 0.0, P*0.02])
    Hc = np.array([H*(1-vacc), 0.0, 0.0, H*vacc])
    H_inc = np.zeros(WEEKS); M_I_ts = np.zeros(WEEKS); P_I_ts = np.zeros(WEEKS)
    for t in range(WEEKS):
        rT = briere(np.array([22.4+7.1*np.cos(2*np.pi*(t*7-200)/365.0))])[0]
        eip = max(1.0, 2.0/rT) if rT > 0.05 else 6.0
        for _ in range(sub):
            Mtot = M.sum() + 1.0
            foi_m = psi * (Pc[2]/P + Hc[2]/H)
            foi_p = phi_p * (M[2]/Mtot) * (Pc[0]/P)
            foi_h = phi_h * (M[2]/Mtot) * (Hc[0]/H) * (1-vacc)
            dM = np.array([
                em0*C[t] - muM*M[0] - foi_m*M[0],
                foi_m*M[0] - muM*M[1] - M[1]/eip,
```

Additional File: Simulation Source Code (je_sim)

Mechanistic simulator for Culex / Japanese encephalitis with extreme-precipitation flushing-and-rebound

Page 3 of 6

```
M[1]/eip - muM*M[2]])
M = np.clip(M + dM*dt, 0, None)
dP = np.array([
    -foi_p*Pc[0] + import_p*dt,
    foi_p*Pc[0] - Pc[1] - Pc[1],
    Pc[1] - Pc[2]])
Pc = np.clip(Pc + dP*dt, 0, None)
dH = np.array([
    -foi_h*Hc[0],
    foi_h*Hc[0] - Hc[1] - Hc[1]/1.5,
    Hc[1]/1.5 - Hc[2],
    Hc[2]])
Hc = np.clip(Hc + dH*dt, 0, None)
H_inc[t] = max(0.0, foi_h*Hc[0]/H*100000.0)
M_I_ts[t] = M[2]; P_I_ts[t] = Pc[2]
Hc = Hc/Hc.sum()*H
return H_inc, M_I_ts, P_I_ts

# ----- partialling-out helper -----
def po_reg(y, treat, X):
    from numpy.linalg import lstsq
    b1,_,_ = lstsq(X, y, rcond=None); ry = y - X@b1
    b2,_,_ = lstsq(X, treat, rcond=None); rt = treat - X@b2
    b,_,_ = lstsq(rt.reshape(-1,1), ry, rcond=None)
    b = float(b[0]); resid = ry - rt*b
    se = np.sqrt((resid@resid)/(len(ry)-2))/np.std(rt, ddof=1)
    t = b/se; p = 2*(1-0.5*(1+math.erf(abs(t)/math.sqrt(2))))
    return b, t, p

def distributed_lag(inc_pool, epe_pool, X):
    out = {}
    for lag in range(0,9):
        Tv = np.zeros_like(inc_pool)
        n = inc_pool.shape[0]
        Tv[lag:] = epe_pool[:n-lag]
        out[lag] = po_reg(inc_pool, Tv, X)
    return out

# ----- main -----
grid_offsets = rng.normal(0, 0.4, size=(N_GRID,2))
weather = [build_weather(grid_offsets[g]) for g in range(N_GRID)]
Cgrid=[]; Cgrid_raw=[]; Rgrid=[]; epegrid=[]; Tgrid=[]
for g in range(N_GRID):
    Td,Rd,ed = weather[g]
    Tw,Rw,epe_w,C,Craw = culex_index(Td,Rd,ed)
    Cgrid.append(C); Cgrid_raw.append(Craw); Rgrid.append(Rw); epegrid.append(epe_w);
    Tgrid.append(Tw)
Cgrid_smooth=[]; Csmooth_raw=[]
for g in range(N_GRID):
    Td,Rd,ed = weather[g]
    _,_,Cs,Csraw = culex_index(Td,Rd,np.zeros_like(ed))
    Cgrid_smooth.append(Cs); Csmooth_raw.append(Csraw)
```

Additional File: Simulation Source Code (je_sim)

Mechanistic simulator for Culex / Japanese encephalitis with extreme-precipitation flushing-and-rebound

Page 4 of 6

```
vuln = np.clip(0.15 + 0.7*np.linspace(0,1,N_GRID) + rng.normal(0,0.05,N_GRID), 0.05, 0.95)
vuln = vuln/vuln.max()
inc_grid=[]; Mi_grid=[]; Pi_grid=[]
for g in range(N_GRID):
    inc,Mi,Pi = run_seir(Cgrid[g], epegrid[g], vuln[g])
    inc_grid.append(inc); Mi_grid.append(Mi); Pi_grid.append(Pi)

# pooled Culex index + EPE for season-adjusted lag regression (event vs smooth)
wk = (np.arange(WEEKS)%52)/52*2*np.pi
seas = np.column_stack([np.sin(wk), np.cos(wk), np.ones(WEEKS)])
def lag_coefs(series_list, lags=range(7)):
    out={}
    for lag in lags:
        cs=[]
        for g in range(N_GRID):
            tv=np.zeros(WEEKS); tv[lag:]=epegrid[g][:WEEKS-lag].astype(float)
            X=np.column_stack([tv, seas[:,2]])
            b,_,_=lstsq(X, series_list[g], rcond=None)
            cs.append(b[0])
        out[lag]=(float(np.mean(cs)), float(np.std(cs)))
    return out
Dlist = [Cgrid_raw[g]-Csmooth_raw[g] for g in range(N_GRID)] # unnormalized event anomaly
dl_D = lag_coefs(Dlist) # flush (lag0 neg) + rebound (lag1-3 pos)
dl_C = lag_coefs(Cgrid) # event-driven Culex (season-confounded, for
    context)
dl_Csmooth = lag_coefs(Cgrid_smooth) # no-event control

# equity
order = np.argsort(vuln)

# ---- diagnostics: raw season-adjusted EPE-lag coefficients (grid 0) ----
g=0
wk2=(np.arange(WEEKS)%52)/52*2*np.pi
seas2=np.column_stack([np.sin(wk2),np.cos(wk2),np.ones(WEEKS)])
from numpy.linalg import lstsq
for name,series in [{"C",Cgrid[g]},"inc",inc_grid[g]},{"M_I",Mi_grid[g]}]:
    line=f"=== {name} raw EPE-lag coef (grid0) ==="
    for lag in range(0,6):
        tv=np.zeros(WEEKS); tv[lag:]=epegrid[g][:WEEKS-lag].astype(float)
        X=np.column_stack([tv,seas2])
        b,_,_=lstsq(X,series,rcond=None)
        line+=f" lag{lag}:{b[0]:+.3f}"
    print(line)
quint = np.array_split(order,5)
eq = {f"q{i+1}": float(np.mean([inc_grid[g].sum() for g in quint[i]])) for i in range(5)}
attack = [float(inc.sum()) for inc in inc_grid]

results = {
    "weeks": WEEKS, "n_grid": N_GRID,
    "vuln": [float(v) for v in vuln],
    "attack_rate_per100k": attack,
    "peak_inc_per100k": [float(np.max(inc)) for inc in inc_grid],
    "epe_per_grid": [int(e.sum()) for e in epegrid],
```

Additional File: Simulation Source Code (je_sim)

Mechanistic simulator for Culex / Japanese encephalitis with extreme-precipitation flushing-and-rebound

Page 5 of 6

```
"equity_by_quintile": eq,
"dL_Culex_event_anomaly": {str(k):[round(v[0],3),round(v[1],3)] for k,v in dL_D.items()},
"dL_Culex_event": {str(k):[round(v[0],3),round(v[1],3)] for k,v in dL_C.items()},
"dL_Culex_smooth_control": {str(k):[round(v[0],3),round(v[1],3)] for k,v in
    dL_Csmooth.items()},
}
with open("_je_results.json","w") as f: json.dump(results,f,indent=2)
print("Attack rate per100k by grid:", [round(x,1) for x in attack])
print("Peak inc per100k:", [round(x,1) for x in results["peak_inc_per100k"]])
print("Equity quintile (Q1 low->Q5 high):", {k:round(v,1) for k,v in eq.items()})
print("Culex event ANOMALY dL (D=C event-C smooth, mean±sd):",
    {k:[round(v[0],3),round(v[1],3)] for k,v in dL_D.items()})
print("Culex dL EVENT (confounded):", {k:[round(v[0],3),round(v[1],3)] for k,v in
    dL_C.items()})
print("Culex dL SMOOTH control:", {k:[round(v[0],3),round(v[1],3)] for k,v in
    dL_Csmooth.items()})

# ----- figures -----
# 1 weather (grid0, year 2018 = weeks 156..208)
g=0
yr0=156; yr1=208
fig,ax=plt.subplots(2,1,figsize=(9,5))
ax[0].plot(range(yr0,yr1), Tgrid[g][yr0:yr1], color="#1f77b4")
ax[0].set_ylabel("Temp (C)"); ax[0].set_title("Synthetic Guangzhou weather (grid 0, 2018)")
ax[1].plot(range(yr0,yr1), Rgrid[g][yr0:yr1], color="#2c7fb8")
edays = np.where(epegrid[g][yr0*7:yr1*7])[0]
ax[1].scatter(np.array(edays)//7+yr0, Rgrid[g][yr0:yr1][np.array(edays)//7], color="red",
    s=12, label="EPE day")
ax[1].set_ylabel("Precip (mm/wk)"); ax[1].legend()
plt.tight_layout(); plt.savefig("_fig_weather.png", dpi=150); plt.close()

# 2 Culex abundance + EPE, and C vs R scatter
fig,ax=plt.subplots(1,2,figsize=(11,4))
ax[0].plot(range(WEEKS), Cgrid[g], color="#4444")
ax[0].scatter(np.where(epegrid[g])[0], Cgrid[g][epegrid[g]], color="red", s=14, label="EPE
    week")
ax[0].set_title("Culex PFI-X index (dip+rebound at EPE)"); ax[0].set_xlabel("week");
    ax[0].set_ylabel("C (norm)"); ax[0].legend()
from numpy.random import default_rng
rr=default_rng(1)
idx=rr.choice(WEEKS,400,replace=False)
ax[1].scatter(Rgrid[g][idx], Cgrid[g][idx], s=6, alpha=0.4, color="#2c7fb8")
ax[1].set_title("C vs weekly precipitation (hump)"); ax[1].set_xlabel("weekly precip mm");
    ax[1].set_ylabel("C")
plt.tight_layout(); plt.savefig("_fig_abundance.png", dpi=150); plt.close()

# 3 incidence with EPE markers
fig,ax=plt.subplots(figsize=(9,4))
ax.plot(range(WEEKS), inc_grid[g], color="#b2182b")
epew=np.where(epegrid[g])[0]
ax.scatter(epew, inc_grid[g][epew], color="black", s=14, label="EPE week")
ax.set_title("Weekly JE incidence per 100k (grid 0)"); ax.set_xlabel("week");
    ax.set_ylabel("inc/100k"); ax.legend()
plt.tight_layout(); plt.savefig("_fig_incidence.png", dpi=150); plt.close()

# 4 equity
fig,ax=plt.subplots(figsize=(6,4))
qs=list(eq.keys()); vals=[eq[k] for k in qs]
ax.bar(qs, vals, color="#762a83")
ax.set_title("Cumulative attack rate by vulnerability quintile");
    ax.set_ylabel("attack/100k")
plt.tight_layout(); plt.savefig("_fig_equity.png", dpi=150); plt.close()
```

Additional File: Simulation Source Code (je_sim)

Mechanistic simulator for Culex / Japanese encephalitis with extreme-precipitation flushing-and-rebound

Page 6 of 6

```
# 5 distributed lag (Culex event anomaly: flush at lag0, rebound lag1-3)
fig,ax=plt.subplots(figsize=(8,4))
lags=list(range(7))
de=[dl_D[l][0] for l in lags]
x=np.arange(len(lags))
colors=["#d73027" if l==0 else "#1f78b4" for l in lags]
ax.bar(x, de, color=colors)
ax.axhline(0,color="k",lw=0.8)
ax.set_xticks(x); ax.set_xticklabels([f"lag{l}" for l in lags])
ax.set_title("Event-induced Culex anomaly by lag (flush at lag 0, rebound at lag 1-3)")
ax.set_xlabel("lag (weeks)"); ax.set_ylabel("anomaly coef")
plt.tight_layout(); plt.savefig("_fig_dlag.png", dpi=150); plt.close()
print("figures saved")
```