Supplementary Material (Online Resource 1) — Python Code for Data Extraction and Estimation

Manuscript ijfs-4502589 — "Pix and the Nowcasting of Brazilian Economic Activity" — International Journal of Financial Studies (MDPI)

This supplementary file accompanies the manuscript and reproduces, for transparency and replicability purposes, the three Python scripts used to (i) extract the IBC-Br, Selic, Pix, IPCA, and IBC-Br Indústria series from the Central Bank of Brazil's open-data APIs, (ii) correct the cumulative-query behavior of the Pix API's reference-period filter and aggregate the series to a monthly total, and (iii) estimate the predictive regressions, out-of-sample tests, and specification-sensitivity checks reported in Sections 3 and 4 of the manuscript.

1. Data extraction (IBC-Br, Selic, IPCA, industrial production, and Pix)

Script used to extract the IPCA (SGS 433) and IBC-Br Indústria (SGS 29603) series, and the complete Pix transaction panel, from the Central Bank of Brazil's SGS and Olinda (EstatisticasTransacoesPix) APIs. Requirements: pip install pandas requests.

```
import requests
import pandas as pd
from datetime import datetime
import time

# ============================================================
# PARTE 1: IPCA e Produção Industrial (API SGS - simples, sem
# o bug de deduplicacao do Pix). Mesma funcao get_sgs() do seu
# script original.
# ============================================================
def get_sgs(codigo, data_inicial=None, data_final=None, timeout=60):
    url = f"https://api.bcb.gov.br/dados/serie/bcdata.sgs.{codigo}/dados?formato=json"
    if data_inicial:
        url += f"&dataInicial={data_inicial}&dataFinal={data_final}"
    r = requests.get(url, timeout=timeout)
    r.raise_for_status()
    df = pd.DataFrame(r.json())
    df["data"] = pd.to_datetime(df["data"], dayfirst=True)
    df["valor"] = df["valor"].astype(float)
    return df

print("Buscando IPCA (SGS 433, variacao % mensal) ...")
ipca = get_sgs(433)
ipca.to_csv("dados_ipca_2020_2026.csv", index=False)
print(f"  OK: {len(ipca)} linhas salvas em dados_ipca_2020_2026.csv")
print(f"  Confira se os valores parecem taxas mensais de inflacao (tipicamente entre -
0.5 e 1.5):")
print(ipca.tail(5))

print("\nBuscando Producao Industrial (SGS 29603, IBC-Br Industria, sem ajuste
sazonal) ...")
try:
    industria = get_sgs(29603)
    industria.to_csv("dados_industria_2020_2026.csv", index=False)
    print(f"  OK: {len(industria)} linhas salvas em dados_industria_2020_2026.csv")
    print(industria.tail(5))
except Exception as e:
    print(f"  FALHOU: {e}")
```

```
    print("  Se der erro, o codigo 29603 pode ter mudado -- me avise que eu confirmo
de novo.")

# ============================================================
# PARTE 2: Pix (API Olinda/OData) - loop mes a mes, resiliente
# a timeouts e erros 500/504 do servidor do BC, com dedup e
# agregacao mensal ao final.
# ============================================================
def get_pix_month(yyyymm, timeout=240, tentativas=3):
    base =
"https://olinda.bcb.gov.br/olinda/servico/Pix_DadosAbertos/versao/v1/odata/"
    url = (f"{base}EstatisticasTransacoesPix(Database=@Database)"
           f"?$format=json&@Database='{yyyymm}'")
    for t in range(tentativas):
        try:
            r = requests.get(url, timeout=timeout)
            r.raise_for_status()
            return pd.DataFrame(r.json()["value"])
        except requests.exceptions.RequestException as e:
            print(f"    tentativa {t+1}/{tentativas} falhou: {e}")
            time.sleep(5)
    return None

print("\n" + "="*60)
print("Iniciando extracao do Pix (pode levar varios minutos) ...")
print("="*60)

meses = pd.period_range("2020-11", datetime.today(), freq="M")
pix_frames = []
meses_com_falha = []

for m in meses:
    yyyymm = m.strftime("%Y%m")
    print(f"Buscando {yyyymm} ...")
    df_mes = get_pix_month(yyyymm)
    if df_mes is None:
        print(f"  FALHOU (todas as tentativas): {yyyymm}")
        meses_com_falha.append(yyyymm)
    else:
        print(f"  OK: {yyyymm} ({len(df_mes)} linhas)")
        pix_frames.append(df_mes)

if not pix_frames:
    raise RuntimeError("Nenhum mes do Pix retornou dados. Confira a conexao/URL.")

pix_raw = pd.concat(pix_frames, ignore_index=True)
print(f"\nLinhas brutas totais do Pix (com duplicacao esperada): {len(pix_raw)}")
pix_raw.to_csv("pix_bruto_v5.csv", index=False)

pix_dedup = pix_raw.drop_duplicates()
print(f"Linhas apos remover duplicatas: {len(pix_dedup)}")
print("\nColunas disponiveis no Pix:", pix_dedup.columns.tolist())

col_mes = "Database"
col_quantidade = "quantidade_total"
col_valor = "valor_total"

faltantes = [c for c in (col_mes, col_quantidade, col_valor) if c not in
pix_dedup.columns]
if faltantes:
    print(f"\nATENCAO: as colunas {faltantes} nao existem no retorno da API do Pix.")
    print("Ajuste os nomes col_mes / col_quantidade / col_valor acima")
    print("usando a lista de colunas impressa mais acima e rode de novo.")
else:
    pix_mensal = (
        pix_dedup
        .groupby(col_mes, as_index=False)[[col_quantidade, col_valor]]
        .sum()
```

```
        .rename(columns={col_mes: "AnoMes",
                         col_quantidade: "quantidade_total",
                         col_valor: "valor_total"})
        .sort_values("AnoMes")
    )
    pix_mensal.to_csv("pix_mensal_agregado.csv", index=False)
    print(f"\npix_mensal_agregado.csv salvo com {len(pix_mensal)} meses.")
    print(pix_mensal.head())
    print(pix_mensal.tail())

if meses_com_falha:
    print(f"\nATENCAO: estes meses do Pix falharam em todas as tentativas:")
    print(meses_com_falha)
    print("Rode de novo mais tarde so para esses meses, se precisar.")

print("\n" + "="*60)
print("CONCLUIDO. Arquivos gerados nesta pasta:")
print("  dados_ipca_2020_2026.csv")
print("  dados_industria_2020_2026.csv")
print("  pix_bruto_v5.csv")
print("  pix_mensal_agregado.csv  <- este e o principal")
print("="*60)
```

2. Duplicate correction and monthly aggregation of Pix data

Script that removes duplicate records generated by the cumulative-query behavior of the Pix API's reference-period filter parameter (described in Section 3.1 of the manuscript) and aggregates transactions into a monthly total series (quantidade_total, valor_total). Requirements: pip install pandas.

```
import pandas as pd

print("Lendo pix_bruto_v5.csv (pode demorar um pouco, arquivo grande) ...")
pix_raw = pd.read_csv("pix_bruto_v5.csv")
print(f"Linhas lidas: {len(pix_raw)}")

pix_dedup = pix_raw.drop_duplicates()
print(f"Linhas apos deduplicar: {len(pix_dedup)}")

pix_mensal = (
    pix_dedup
    .groupby("AnoMes", as_index=False)[["QUANTIDADE", "VALOR"]]
    .sum()
    .rename(columns={"QUANTIDADE": "quantidade_total", "VALOR": "valor_total"})
    .sort_values("AnoMes")
)

pix_mensal.to_csv("pix_mensal_agregado.csv", index=False)
print(f"\npix_mensal_agregado.csv salvo com {len(pix_mensal)} meses.")
print(pix_mensal.head())
print(pix_mensal.tail())

meses_esperados = set(pd.period_range("2020-11", pd.Timestamp.today(),
freq="M").strftime("%Y%m"))
meses_obtidos = set(pix_mensal["AnoMes"].astype(str))
faltando = sorted(meses_esperados - meses_obtidos)
if faltando:
    print(f"\nMeses ainda faltando: {faltando}")
```

3. Estimation — predictive regressions, out-of-sample tests, and robustness checks (Sections 3–4)

Script that produces: the baseline full-sample regressions (Naive, AR(1), reduced and full predictive regressions with Pix); the expanding-window out-of-sample forecasts and the Diebold-Mariano and Clark-West (2007) tests comparing them; the seasonal robustness checks with monthly and quarterly dummy specifications; the additional controls for IPCA and industrial production; the Zivot-Andrews structural-break test applied to the Pix growth series; the residual diagnostics (Durbin-Watson, Breusch-Godfrey, Breusch-Pagan, Augmented Dickey-Fuller on residuals, Newey-West HAC standard errors, and Cook's distance); and the first-differenced specification. Requirements: pip install pandas numpy statsmodels scipy.

```
"""
Estimação completa - Pix como indicador de nowcasting do IBC-Br
Reproduz os resultados reportados nas Secoes 3 e 4 do manuscrito
"Pix and the Nowcasting of Brazilian Economic Activity" (IJFS, ijfs-4502589).

Entradas esperadas (mesma pasta):
  - pix_mensal_agregado.csv   (colunas: AnoMes, quantidade_total, valor_total)
  - Serie_433.csv             (IBC-Br, Selic, IPCA e IBC-Br Industria consolidados)

Requer: pip install pandas numpy statsmodels
"""

import pandas as pd
import numpy as np
import statsmodels.api as sm
from statsmodels.stats.stattools import durbin_watson
from statsmodels.stats.diagnostic import acorr_breusch_godfrey, het_breuschpagan
from statsmodels.tsa.stattools import adfuller
from statsmodels.tsa.stattools import zivot_andrews
from statsmodels.stats.outliers_influence import OLSInfluence

pd.set_option("display.width", 140)

# ============================================================
# 1. LEITURA E PREPARACAO DOS DADOS
# ============================================================
pix = pd.read_csv("pix_mensal_agregado.csv")
pix["data"] = pd.to_datetime(pix["AnoMes"].astype(str), format="%Y%m")
pix = pix.sort_values("data").reset_index(drop=True)
pix["pix_valor_growth"] = pix["valor_total"].pct_change()

serie = pd.read_csv("Serie_433.csv")
serie["data"] = pd.to_datetime(serie["data"])
serie = serie.sort_values("data").reset_index(drop=True)
# Colunas esperadas em Serie_433.csv: ibcbr, selic, ipca, ibcbr_industria
# (ajustar os nomes abaixo caso o cabecalho do arquivo real seja diferente)
df = serie.merge(pix[["data", "pix_valor_growth"]], on="data", how="inner")
df["ibcbr_growth"] = df["ibcbr"].pct_change()
df["ibcbr_growth_l1"] = df["ibcbr_growth"].shift(1)
df["selic_l1"] = df["selic"].shift(1)
df["pix_growth_l1"] = df["pix_valor_growth"].shift(1)
df["ipca_l1"] = df["ipca"].shift(1) if "ipca" in df.columns else np.nan
df["industria_growth"] = df["ibcbr_industria"].pct_change() if "ibcbr_industria" in
df.columns else np.nan
df["industria_growth_l1"] = df["industria_growth"].shift(1)
df["mes"] = df["data"].dt.month
df["trimestre"] = df["data"].dt.quarter
df = df.dropna(subset=["ibcbr_growth", "ibcbr_growth_l1", "selic_l1",
"pix_growth_l1"]).reset_index(drop=True)

print(f"Amostra final: {len(df)} observacoes, {df['data'].min():%Y-%m} a
{df['data'].max():%Y-%m}")
```

```
# ============================================================
# 2. MODELOS DE BASE (Tabelas 5-8: regressao em amostra completa)
# ============================================================
y = df["ibcbr_growth"]

# Modelo 1 - Naive (passeio aleatorio): usado apenas na comparacao fora da amostra
# Modelo 2 - AR(1)
X_ar1 = sm.add_constant(df[["ibcbr_growth_l1"]])
m_ar1 = sm.OLS(y, X_ar1).fit()

# Modelo 3 - Bridge reduzida (AR(1) + Selic)
X_reduzida = sm.add_constant(df[["ibcbr_growth_l1", "selic_l1"]])
m_reduzida = sm.OLS(y, X_reduzida).fit()

# Modelo 4/5 - Bridge completa (AR(1) + Selic + Pix)
X_completa = sm.add_constant(df[["ibcbr_growth_l1", "selic_l1", "pix_growth_l1"]])
m_completa = sm.OLS(y, X_completa).fit()

print("\n=== Modelo Bridge completa (base) ===")
print(m_completa.summary())

# ============================================================
# 3. PREVISAO FORA DA AMOSTRA - JANELA EXPANSIVA (Tabela 9)
# ============================================================
def previsao_expansiva(df, especificacao, janela_minima=24):
    """Reestimacao com janela expansiva. `especificacao` e uma lista de
    colunas de regressores (sem a constante); passar [] para o modelo Naive."""
    erros = []
    for t in range(janela_minima, len(df)):
        treino = df.iloc[:t]
        alvo_real = df["ibcbr_growth"].iloc[t]
        if especificacao == "naive":
            previsto = treino["ibcbr_growth"].iloc[-1]
        else:
            X_treino = sm.add_constant(treino[especificacao])
            modelo_t = sm.OLS(treino["ibcbr_growth"], X_treino).fit()
            X_teste = sm.add_constant(
                df[especificacao].iloc[[t]], has_constant="add"
            )
            previsto = modelo_t.predict(X_teste).iloc[0]
        erros.append(alvo_real - previsto)
    return np.array(erros)

erros_naive = previsao_expansiva(df, "naive")
erros_ar1 = previsao_expansiva(df, ["ibcbr_growth_l1"])
erros_reduzida = previsao_expansiva(df, ["ibcbr_growth_l1", "selic_l1"])
erros_completa = previsao_expansiva(df, ["ibcbr_growth_l1", "selic_l1",
"pix_growth_l1"])

def rmse(e):
    return float(np.sqrt(np.mean(e ** 2)))

print("\n=== RMSE fora da amostra (janela expansiva) ===")
for nome, e in [("Naive", erros_naive), ("AR(1)", erros_ar1),
                ("Bridge reduzida", erros_reduzida), ("Bridge completa",
erros_completa)]:
    print(f"  {nome}: RMSE = {rmse(e):.5f}  (n={len(e)})")

# ============================================================
# 4. TESTE DIEBOLD-MARIANO (comparacao nao aninhada de acuracia)
# ============================================================
def diebold_mariano(e1, e2, h=1):
    d = e1 ** 2 - e2 ** 2
    dbar = d.mean()
    T = len(d)
    gamma0 = np.var(d, ddof=0)
    dm_stat = dbar / np.sqrt(gamma0 / T)
```

```
    from scipy import stats
    p_valor = 2 * (1 - stats.norm.cdf(abs(dm_stat)))
    return dm_stat, p_valor

dm_stat, dm_p = diebold_mariano(erros_reduzida, erros_completa)
print(f"\nDiebold-Mariano (Bridge reduzida vs. completa): stat={dm_stat:.3f},
p={dm_p:.3f}")

# ============================================================
# 5. TESTE CLARK-WEST (modelos aninhados) - Tabela 10
# ============================================================
def clark_west(erros_pequeno, erros_grande, previstos_pequeno, previstos_grande):
    """Clark & West (2007), teste de acuracia preditiva para modelos aninhados."""
    f = (erros_pequeno ** 2) - (erros_grande ** 2) + (previstos_pequeno -
previstos_grande) ** 2
    T = len(f)
    fbar = f.mean()
    se = f.std(ddof=1) / np.sqrt(T)
    cw_stat = fbar / se
    from scipy import stats
    p_valor = 1 - stats.norm.cdf(cw_stat)  # teste unilateral
    return cw_stat, p_valor

# (obter tambem os valores previstos, nao so os erros, para o termo de ajuste do CW)
def previsao_expansiva_com_previstos(df, especificacao, janela_minima=24):
    previstos = []
    for t in range(janela_minima, len(df)):
        treino = df.iloc[:t]
        if especificacao == "naive":
            p = treino["ibcbr_growth"].iloc[-1]
        else:
            X_treino = sm.add_constant(treino[especificacao])
            modelo_t = sm.OLS(treino["ibcbr_growth"], X_treino).fit()
            X_teste = sm.add_constant(df[especificacao].iloc[[t]], has_constant="add")
            p = modelo_t.predict(X_teste).iloc[0]
        previstos.append(p)
    return np.array(previstos)

prev_reduzida = previsao_expansiva_com_previstos(df, ["ibcbr_growth_l1", "selic_l1"])
prev_completa = previsao_expansiva_com_previstos(df, ["ibcbr_growth_l1", "selic_l1",
"pix_growth_l1"])

cw_stat, cw_p = clark_west(erros_reduzida, erros_completa, prev_reduzida,
prev_completa)
print(f"Clark-West (Bridge reduzida vs. completa com Pix): stat={cw_stat:.3f},
p={cw_p:.3f}")

# ============================================================
# 6. ROBUSTEZ - DUMMIES SAZONAIS (mensais e trimestrais) - Tabela 11
# ============================================================
dummies_mes = pd.get_dummies(df["mes"], prefix="mes", drop_first=True).astype(float)
X_sazonal_mensal = sm.add_constant(
    pd.concat([df[["ibcbr_growth_l1", "selic_l1", "pix_growth_l1"]], dummies_mes],
axis=1)
)
m_sazonal_mensal = sm.OLS(y, X_sazonal_mensal).fit()

dummies_trim = pd.get_dummies(df["trimestre"], prefix="trim",
drop_first=True).astype(float)
X_sazonal_trim = sm.add_constant(
    pd.concat([df[["ibcbr_growth_l1", "selic_l1", "pix_growth_l1"]], dummies_trim],
axis=1)
)
m_sazonal_trim = sm.OLS(y, X_sazonal_trim).fit()

print("\n=== Robustez sazonal ===")
print(f"  Dummies mensais    - coef. Pix:
{m_sazonal_mensal.params['pix_growth_l1']:.5f} "
      f"(p={m_sazonal_mensal.pvalues['pix_growth_l1']:.3f})")
```

```
print(f"  Dummies trimestrais- coef. Pix: {m_sazonal_trim.params['pix_growth_l1']:.5f}
"
      f"(p={m_sazonal_trim.pvalues['pix_growth_l1']:.3f})")

# ============================================================
# 7. CONTROLES ADICIONAIS - IPCA e producao industrial - Tabela 12
# ============================================================
if df["ipca_l1"].notna().any():
    X_ipca = sm.add_constant(df[["ibcbr_growth_l1", "selic_l1", "pix_growth_l1",
"ipca_l1"]].dropna())
    m_ipca = sm.OLS(y.loc[X_ipca.index], X_ipca).fit()
    print(f"\n  Controle IPCA      - coef. Pix: {m_ipca.params['pix_growth_l1']:.5f} "
          f"(p={m_ipca.pvalues['pix_growth_l1']:.3f})")

if df["industria_growth_l1"].notna().any():
    X_ind = sm.add_constant(
        df[["ibcbr_growth_l1", "selic_l1", "pix_growth_l1",
"industria_growth_l1"]].dropna()
    )
    m_ind = sm.OLS(y.loc[X_ind.index], X_ind).fit()
    print(f"  Controle Industria - coef. Pix: {m_ind.params['pix_growth_l1']:.5f} "
          f"(p={m_ind.pvalues['pix_growth_l1']:.3f})")

# ============================================================
# 8. QUEBRA ESTRUTURAL - ZIVOT-ANDREWS (serie de crescimento do Pix)
# ============================================================
za_stat, za_p, za_crit, za_lag, za_bp = zivot_andrews(df["pix_valor_growth"].dropna(),
regression="c")
print(f"\n=== Zivot-Andrews (crescimento do Pix) ===")
print(f"  estatistica={za_stat:.3f}, p-valor={za_p:.3f}, ponto de quebra estimado
(indice)={za_bp}")

# ============================================================
# 9. DIAGNOSTICO DE RESIDUOS DO MODELO BASE - Tabela 13
# ============================================================
resid = m_completa.resid

dw = durbin_watson(resid)
bg_stat, bg_p, _, _ = acorr_breusch_godfrey(m_completa, nlags=4)
bp_stat, bp_p, _, _ = het_breuschpagan(resid, m_completa.model.exog)
adf_stat, adf_p, *_ = adfuller(resid, regression="c", autolag="AIC")

print("\n=== Diagnostico de residuos (modelo base) ===")
print(f"  Durbin-Watson: {dw:.3f}")
print(f"  Breusch-Godfrey (4 defasagens): stat={bg_stat:.3f}, p={bg_p:.3f}")
print(f"  Breusch-Pagan: stat={bp_stat:.3f}, p={bp_p:.3f}")
print(f"  ADF nos residuos: stat={adf_stat:.3f}, p={adf_p:.3f}")

# Erros-padrao HAC de Newey-West (robustos a autocorrelacao e heterocedasticidade)
m_completa_hac = sm.OLS(y, X_completa).fit(cov_type="HAC", cov_kwds={"maxlags": 4})
print(f"\n  Coef. Pix (EP convencional): {m_completa.params['pix_growth_l1']:.5f} "
      f"(p={m_completa.pvalues['pix_growth_l1']:.3f})")
print(f"  Coef. Pix (EP HAC Newey-West): {m_completa_hac.params['pix_growth_l1']:.5f}
"
      f"(p={m_completa_hac.pvalues['pix_growth_l1']:.3f})")

# Distancia de Cook (observacoes influentes)
influencia = OLSInfluence(m_completa)
cooks_d = influencia.cooks_distance[0]
limite_cook = 4 / len(df)
obs_influentes = df.loc[cooks_d > limite_cook, "data"]
print(f"\n  Observacoes com distancia de Cook acima do limite
(4/n={limite_cook:.4f}):")
print(f"  {list(obs_influentes.dt.strftime('%Y-%m'))}")

# ============================================================
# 10. MODELO EM PRIMEIRAS DIFERENCAS - Tabela 14
# ============================================================
```

```
df_diff = df.copy()
df_diff["d_ibcbr_growth"] = df_diff["ibcbr_growth"].diff()
df_diff["d_ibcbr_growth_l1"] = df_diff["ibcbr_growth_l1"].diff()
df_diff["d_selic_l1"] = df_diff["selic_l1"].diff()
df_diff["d_pix_growth_l1"] = df_diff["pix_growth_l1"].diff()
df_diff = df_diff.dropna(
    subset=["d_ibcbr_growth", "d_ibcbr_growth_l1", "d_selic_l1", "d_pix_growth_l1"]
)

X_diff = sm.add_constant(df_diff[["d_ibcbr_growth_l1", "d_selic_l1",
"d_pix_growth_l1"]])
m_diff = sm.OLS(df_diff["d_ibcbr_growth"], X_diff).fit()
m_diff_hac = sm.OLS(df_diff["d_ibcbr_growth"], X_diff).fit(cov_type="HAC",
cov_kwds={"maxlags": 4})

adf_diff_stat, adf_diff_p, *_ = adfuller(m_diff.resid, regression="c", autolag="AIC")

print("\n=== Modelo em primeiras diferencas ===")
print(f"  Coef. Pix (EP convencional): {m_diff.params['d_pix_growth_l1']:.5f} "
      f"(p={m_diff.pvalues['d_pix_growth_l1']:.3f})")
print(f"  Coef. Pix (EP HAC):          {m_diff_hac.params['d_pix_growth_l1']:.5f} "
      f"(p={m_diff_hac.pvalues['d_pix_growth_l1']:.3f})")
print(f"  ADF nos residuos (diferencas): stat={adf_diff_stat:.3f},
p={adf_diff_p:.3f}")

print("\nCONCLUIDO. Confira cada valor impresso acima contra as Tabelas 5 a 14 do
manuscrito.")
```