E.T.S. de Ingeniería Industrial,
Informática y de Telecomunicación

Validación de la herramienta de linealización de FAST: Análisis temporal y frecuencial en lazo abierto y cerrado

Máster Universitario en
Ingeniería Industrial

Trabajo Fin de Máster

Idoia Lizarraga Zubeldia
Jorge Elso Torralba
Pamplona, 27 de septiembre de 2019

upna Universidad Pública de Navarra
Nafarroako Unibertsitate Publikoa

Resumen

El presente Trabajo Fin de Máster ha sido realizado en el grupo de Control de Aerogeneradores del departamento de Ingeniería de la Universidad Pública de Navarra con el objetivo de validar la herramienta de linealización del software de libre acceso FAST desarrollado por el *National Renewable Energy Laboratory*. Para llevar a cabo dicha tarea se han seguido los siguientes pasos:

- Obtener el modelo lineal de todas las zonas de funcionamiento del aerogenerador, incluso a bajas velocidades de viento.
- Adaptar y post-procesar las salidas del modelo lineal de FAST para generar un modelo lineal estandarizado compatible con herramientas comerciales.
- Validar el modelo lineal comparándolo con el sistema no lineal simulado con FAST. Para ello se han introducido escalones, rampas y senos tanto en lazo abierto como en lazo cerrado, y se ha realizado la comparativa del análisis frecuencial del sistema en lazo abierto.
- Crear un procedimiento basado en scripts de MATLAB para validar el modelo lineal y su control.

Palabras clave

Aerogenerador, FAST, modelo lineal, validación linealización, lazo cerrado

Abstract

This Master's Degree Final Project has been done in the Wind Turbine Control group of the Engineer department of the Public University of Navarre with the main objective of validating the linearization tool of the public software FAST, developed by the National Renewable Energy Laboratory. In order to fulfill the main goal, the following steps were taken:

- Attainment of the lineal model of the wind turbine for all operating zones, even at low wind speed.
- Adaptation and post-processing of the outputs of FAST linear models to generate a standardized lineal model compatible with commercial tools.
- Validation of the linear model by comparing it with the nonlinear model simulated with FAST. In order to do so, steps, ramps and sins in open and closed loop has been introduced. An open loop frequency analysis has be done as well.
- Creation of a procedure based on MATLAB scripts to validate the linear model and its control.

Key words

Wind turbine, FAST, linear model, validation, linearization, closed loop

Índice

Lista de Figuras

Lista de Tablas

Introducción

En la actualidad, las herramientas de ingeniería asistida por ordenador (CAE) son empleadas en prácticamente todos los sectores tecnológicos. Su uso está ligado a una reducción en costes y tiempo, así como en una mejora de la calidad, durabilidad y conocimiento sobre el producto.

El crecimiento del sector eólico está vinculado al avance de la tecnología, no sólo de fabricación y de materiales, sino que también de las herramientas CAE. El software libre líder en el sector es FAST, acrónimo de *Fatigue Aerodynamics Structures Turbulence*. Dicha herramienta ha sido desarrollada por el *National Renewable Energy Laboratory* (NREL) y no sólo permite simular la respuesta dinámica del comportamiento de un aerogenerador, sino que también permite obtener un modelo lineal de la máquina que permite, entre otras cosas, diseñar controladores que la gobiernen.

En previos proyectos realizados en la Universidad Pública de Navarra, en el grupo de Control de Aerogeneradores y en estrecha vinculación con el departamento de control de Siemens Gamesa Renewable Energy, se ha diseñado el control y analizado la máquina de NREL de 5 MW [1][2]. Para ello, se ha partido de linealizaciones obtenidas con FAST. Sin embargo, en ninguno de dichos proyectos se ha prestado especial atención a la herramienta de linealización de FAST, sino a su función como simulador.

Puesto que FAST ofrece la posibilidad de linealizar y simular la misma máquina, la cuestión planteada en este proyecto es la validación de la linealización, es decir, verificar su fidelidad al sistema simulado.

Hasta el momento, en la bibliografía del control de aerogeneradores no se encuentran linealizaciones del sistema a velocidades bajas de viento, ya que debido al buen funcionamiento de las estrategias no lineales, no se han considerado de interés. Sin embargo, interesa obtener el modelo lineal del sistema en todas las zonas de operación del aerogenerador para poder diseñar los controladores y analizar el comportamiento del sistema en todo el rango de velocidades de viento. Por ello, en el presente proyecto se va a linealizar, diseñar y simular el comportamiento del aerogenerador no solo en zona nominal, sino en todas las zonas de operación de la turbina eólica.

Objetivos

El objetivo principal de este Trabajo Fin de Máster es la validación de la herramienta de linealización que ofrece el software de libre acceso FAST y la modificación de su salida para compatibilizarla con herramientas post-procesadoras de diseño. La estructura del trabajo se ha diseñado de acuerdo con el cumplimiento de los siguientes subobjetivos:

- Estudiar el modelo matemático y los fundamentos teóricos de un aerogenerador.
- Aprender a trabajar con el software FAST tanto como herramienta de linealización como simulador.
- Obtener el modelo lineal de todas las zonas de funcionamiento de un aerogenerador.
- Adaptar y post-procesar las salidas del modelo lineal de FAST para generar un modelo lineal con una estructura estandarizada y compatible con post-procesadores comerciales.
- Validar el modelo lineal obtenido comparándolo con el sistema simulado con FAST. Para ello se deben realizar comparativas en lazo abierto y en lazo cerrado tanto con entradas

escalón como con entradas rampa o sinusoidales. También se debe comparar el comportamiento frecuencial empleando los diagramas de Bode.

- Crear un procedimiento de validación del modelo lineal y su control mediante el uso de herramientas creadas en MATLAB (scripts).

Además, con el propósito de que este proyecto sea una herramienta útil para futuros trabajos, se ha prestado especial atención al formato de los programas. De este modo, se ha conseguido un código versátil y fácilmente comprensible para futuros usuarios.

Fundamentos teóricos

Un aerogenerador se define como una máquina empleada para transformar la energía cinética del viento en energía eléctrica. A modo de breve resumen, los principales componentes de un aerogenerador están representados en la Figura 1.

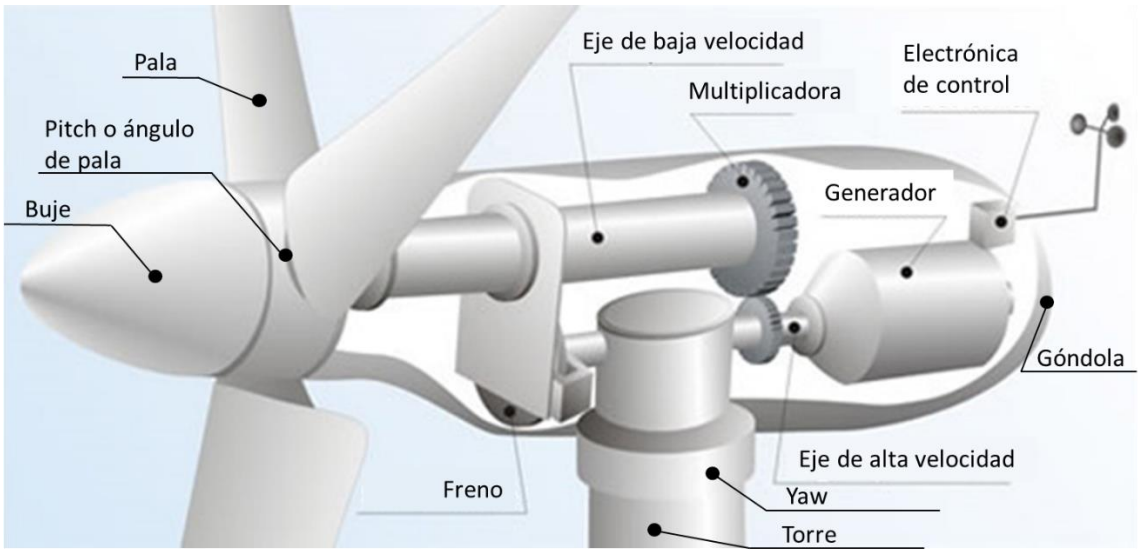

Figura 1. Componentes aerogenerador. Imagen modificada a partir de http://www.daviddarling.info/images2/wind_turbine_nacelle.jpg

El rotor de un aerogenerador está formado por las palas y es el encargado de captar la energía cinética del viento y transformarla en energía rotativa. Al estar las tres palas unidas al eje de baja velocidad a través del buje, la velocidad de rotación de las palas es equivalente a la velocidad del rotor y a la velocidad del eje de baja velocidad.

Debido a una limitación teórica, la energía presente en el viento nunca va a ser transformada en su totalidad en energía mecánica. De acuerdo con el límite de Betz, la máxima potencia que se puede extraer del viento corresponde con el 59.3% de la potencia del viento. La relación entre la potencia extraíble del viento y la potencia del viento se define como el coeficiente de potencia (C_p) y depende del ángulo de paso de la pala o pitch (β) y de la relación de velocidad periférica (TSR[1] o λ).

$$C_p(\beta,\lambda) = \frac{P_{turbina}}{P_{viento}} \tag{1}$$

$$\lambda = TSR = \frac{Velocidad\ de\ la\ periferia\ de\ la\ pala}{Velocidad\ del\ viento} = \frac{R * \omega}{V} \tag{2}$$

$$C_{p,\max} = 0.593 \tag{3}$$

Empleando el coeficiente de potencia se puede obtener la potencia y el par de la turbina eólica.

$$P_{viento} = \frac{1}{2} * \rho * A * V^3 \tag{4}$$

$$P_{turbina} = \frac{1}{2} * \rho * C_p(\beta,\lambda) * A * V^3 \tag{5}$$

[1] TSR: *Tip-Speed-Ratio*

$$T_{turbina} = T_r = \frac{P_{turbina}}{\Omega_r} = \frac{1}{2} * \rho * C_p(\beta, \lambda) * (\pi * R^2) * \frac{V^3}{\Omega_r} \quad (6)$$

Debido a la baja velocidad de giro de las palas, se emplea una multiplicadora previa al generador eléctrico para aumentar la velocidad del eje de rotación a cambio de reducir el par. Esto se debe a que la potencia se puede suponer igual en ambos ejes, es decir, sin pérdidas.

$$\Omega_g = Gbx * \Omega_r \quad (7)$$

$$P = T_r * \Omega_r = T_g * \Omega_g \quad (8)$$

$$T_r = Gbx * T_g \quad (9)$$

Según el tipo de máquina eléctrica empleada como generador, la configuración de la máquina será diferente. Las principales topologías empleadas en aerogeneradores son:

- Máquina asíncrona de jaula de ardilla con conexión directa a red
- Máquina asíncrona doblemente alimentada (MADA o DFIG)
- Máquina asíncrona o máquina síncrona con estructura *Full Converter*

La máquina asíncrona de jaula de ardilla con conexión directa a red fue la primera topología en instalarse pero actualmente está en desuso por su limitación a velocidad fija. Tanto la DFIG como la *Full Converter* permiten trabajar con velocidad variable y en ambos casos la estructura incluye al menos un convertidor electrónico de potencia [3].

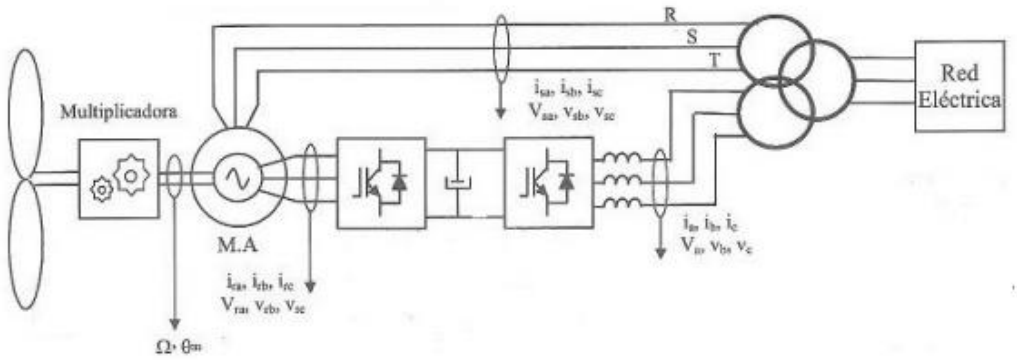

Figura 2. Estructura MADA [3]

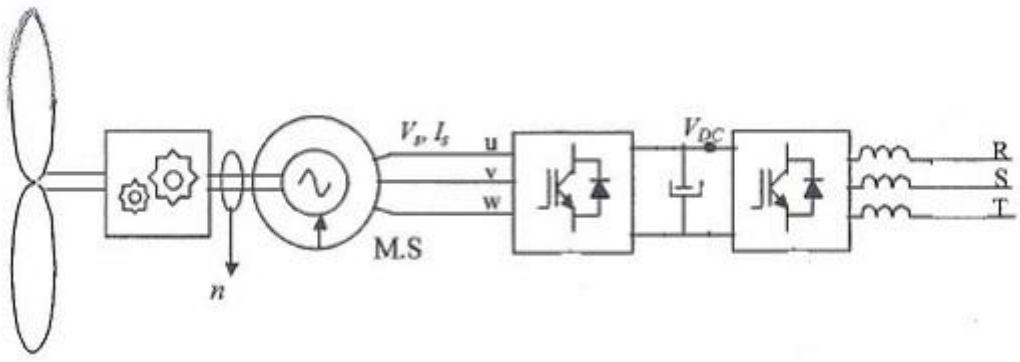

Figura 3. Estructura Full Converter [3]

En ambas estructuras, el control de los convertidores se realiza mediante el encendido y apagado de los semiconductores, lo que se traduce en corrientes y tensiones. Dichas corrientes y tensiones son a su vez potencia y energía que son extraídas del generador eléctrico y vertidas a la red (10).

$$P_{g,elec} = 3 * U * I \text{ [2]} \quad (10)$$

Dicha potencia eléctrica extraída del generador eléctrico se traduce en una fuerza electromotriz que se opone a la rotación del eje rápido del sistema, convirtiendo la potencia mecánica (rotación) en potencia eléctrica.

$$P_{g,elec} = P_{g,mec} * \mu_g \text{ [3]} \quad (11)$$

Empleando la ecuación (8) que relaciona la potencia, el par y la velocidad, se puede reescribir el par del generador eléctrico en función de la tensión y corriente.

$$T_g = \frac{P_{g,elec}}{\Omega_g * \mu_g} = \frac{3 * U * I}{\Omega_g * \mu_g} \quad (12)$$

Por lo tanto, el control del convertidor (corriente y tensión) permite utilizar el par eléctrico como variable de control del aerogenerador.

El objetivo del presente trabajo es el control del aerogenerador, por lo que el funcionamiento del generador y su convertidor pasan a un segundo plano. A efectos prácticos, se van a modelar como un volante de inercia (Figura 4) que recibe un par aerodinámico que puede ser ajustado mediante la acción del pitch y que varía con el viento, y un par eléctrico que se opone y puede manejarse mediante el control de par (14).

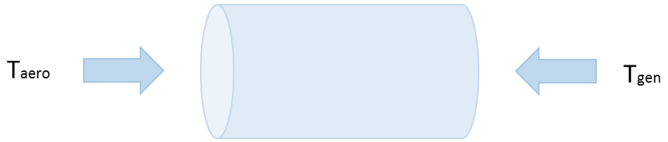

Figura 4. Comportamiento simplificado aerogenerador. Volante de inercia

$$T_{aero}(t) = \frac{1}{2} * \rho * C_p(\beta, \lambda) * \pi * \mathrm{R}^2 * \frac{\mathrm{V(t)}^3}{\Omega_\mathrm{r}(t)} \quad (13)$$

$$T_{aero}(t) - Gbx * T_g(t) = J_{tot} * \dot{\omega}_r(t) \quad (14)$$

La inercia del aerogenerador puede definirse principalmente como la inercia del rotor (eje lento) y la inercia del generador (eje rápido) (15).

$$J_{tot} = J_{rotor} + Gbx^2 * J_{gen} \quad (15)$$

Expresando las anteriores ecuaciones empleando la transformada de Laplace (16), se puede modelar matemáticamente el aerogenerador a partir de bloques de Simulink tal y como se observa en la Figura 5.

$$T_{aero}(s) - Gbx * T_g(s) = J_{tot} * s * \Omega_r(s) \quad (16)$$

[2] Suponiendo que la potencia activa es igual a la potencia aparente, es decir, que el factor de potencia es unitario y no hay potencia reactiva.

[3] Rendimiento (μ_g) inferior a la unidad por pérdidas de potencia debidas a pérdidas eléctricas, magnéticas y mecánicas.

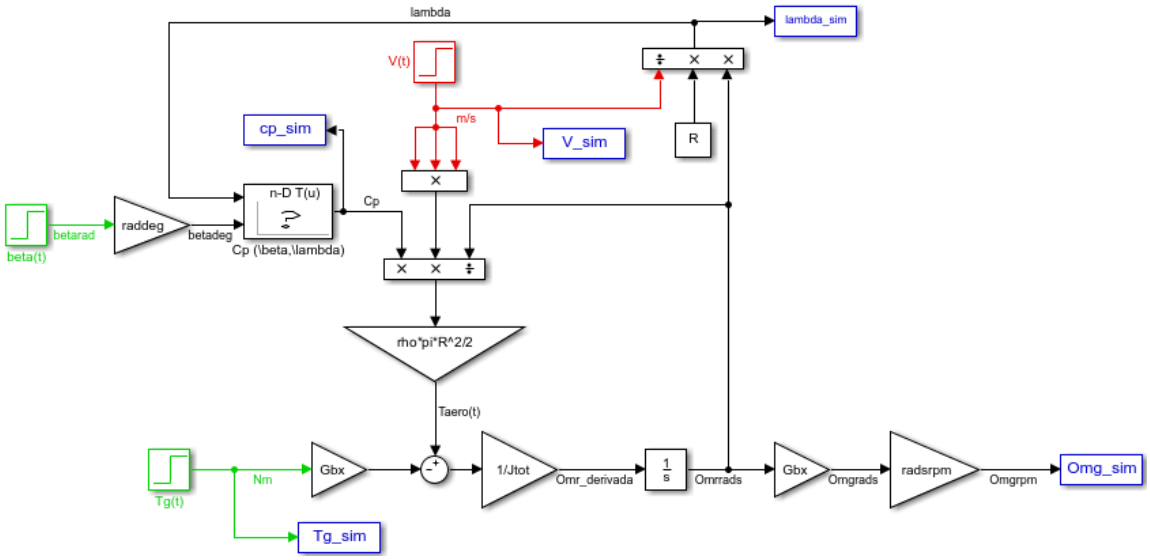

Figura 5. Modelo matemático simplificado con Simulink

Teniendo en cuenta que el viento es una perturbación del sistema ($V(t)$), el control del aerogenerador se realiza por medio del control del par ($T_g(t)$) y del control del paso de pala o pitch ($\beta(t)$). El control del par se realiza por medio de los convertidores, tal y como se ha descrito anteriormente; y el control del paso de pala se realiza por medio de actuadores, ya sean eléctricos o hidráulicos, que modifican el ángulo de paso de la pala.

Figura 6. Ejemplo actuador eléctrico del ángulo de paso.
Fuente: www.konstruktionspraxis.vogel.de/innovative-systemloesung-fuer-die-pitchverstellung-a-143944

A partir del control de par y de pitch, se busca maximizar el coeficiente de potencia del aerogenerador para maximizar la extracción de potencia disponible en el viento y mantener las cargas en niveles admisibles. Sin embargo, a pesar de trabajar con generadores de velocidad variable, se debe operar dentro de un rango limitado de velocidad de rotación. Como consecuencia, a altas velocidades de viento se debe respetar la potencia máxima del generador para evitar dañarlo.

Zonas de operación

Atendiendo a los límites del generador en cuanto a rango de velocidad y potencia, según la velocidad de viento se identifican diferentes zonas de operación.

A partir de la relación de velocidad periférica definida previamente (2), se define la velocidad óptima de giro del rotor (17) y del generador (18) en función de la velocidad de viento.

$$\Omega_{r_{opt}} = \frac{\lambda_{opt} * V}{R} \quad \left(\frac{rad}{s}\right) \tag{17}$$

$$\Omega_{g_{opt}} = Gbx * \Omega_{r_{opt}} = Gbx * \frac{\lambda_{opt} * V}{R} \quad \left(\frac{rad}{s}\right) \tag{18}$$

Reescribiendo el par del generador (19) en función de la velocidad periférica en lugar de la velocidad de viento (20)(21), se define un nuevo parámetro K_{opt} (22) que permite definir el par (23) y la potencia (24) como una relación cuadrática y cúbica con la velocidad de giro del generador.

$$T_g = \frac{P}{\Omega_g} = \frac{\frac{1}{2} * \rho * \pi * R^2 * C_p * V^3}{\Omega_g} \tag{19}$$

$$V = \frac{R * \Omega_r}{\lambda_{opt}} = \frac{R * \Omega_g}{\lambda_{opt} * Gbx} \tag{20}$$

$$T_g = \frac{1}{2} * \rho * \pi * R^2 * C_p * \frac{R^3 * \Omega_g^3}{Gbx^3 * \lambda_{opt}^3 * \Omega_g} \tag{21}$$

$$K_{opt_g} = \frac{1}{2} * \rho * \pi * C_{p,max} * \frac{R^5}{Gbx^3 * \lambda_{opt}^3} \tag{22}$$

$$T_g = K_{opt_g} * \Omega_g^2 \tag{23}$$

$$P = T_g * \Omega_g = K_{opt_g} * \Omega_g^3 \tag{24}$$

Las anteriores ecuaciones no tienen en cuenta los límites del propio generador. A pesar de tratar el generador, el convertidor y a la conexión a red como un bloque, el sistema tiene ciertos límites. El generador está diseñado para girar dentro de un rango de velocidades ($\Omega_{g,min} - \Omega_{nom}$) que permite mantener el deslizamiento de la máquina eléctrica dentro de unos límites aceptables y evitar poner en riesgo su integridad. Además, el generador, dentro de sus características eléctricas, tiene un par electromagnético máximo o nominal ($T_{g.nom}$), que se opone al par mecánico sin dañar el cableado.

Teniendo en cuenta los límites del aerogenerador en cuanto a velocidad de giro ($\Omega_{g,min}$ y Ω_{nom}) y par ($T_{g.nom}$), se define el modelo de la turbina limitada. En las siguientes figuras (Figura 7, Figura 8, Figura 9 y Figura 10) se contrasta el comportamiento de dicha turbina limitada con una turbina teórica que trabaja para todas las velocidades de viento dentro de la zona cuadrática, es decir, maximizando la potencia extraída del viento.

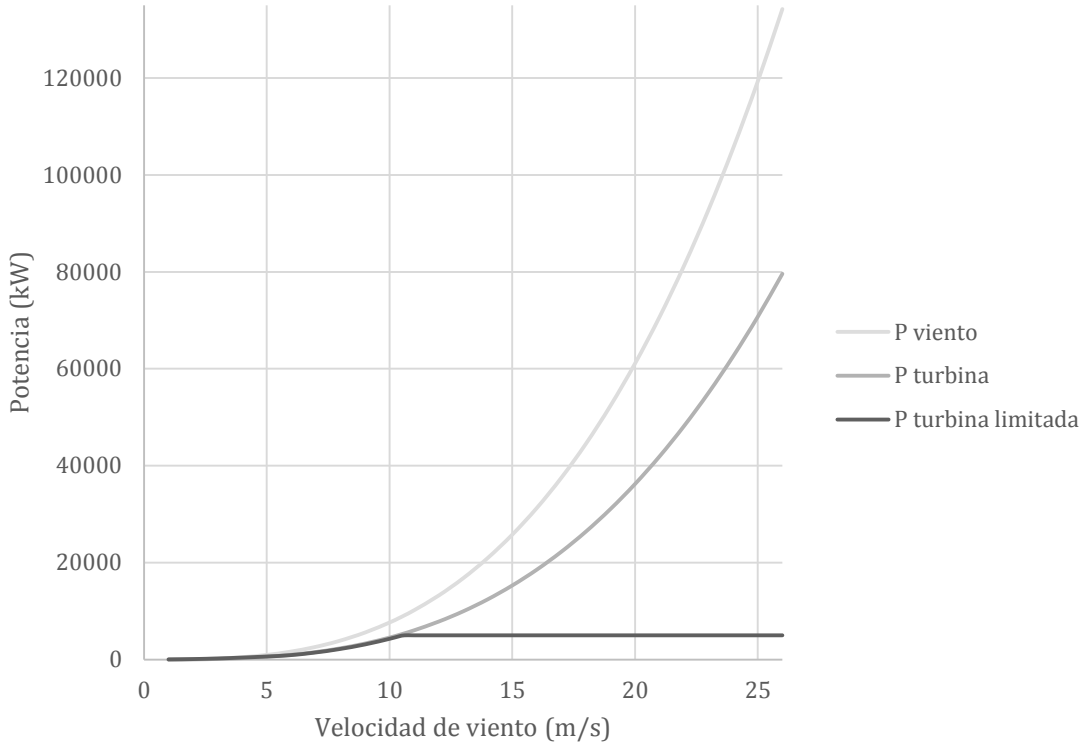

Figura 7. Potencia VS velocidad de viento

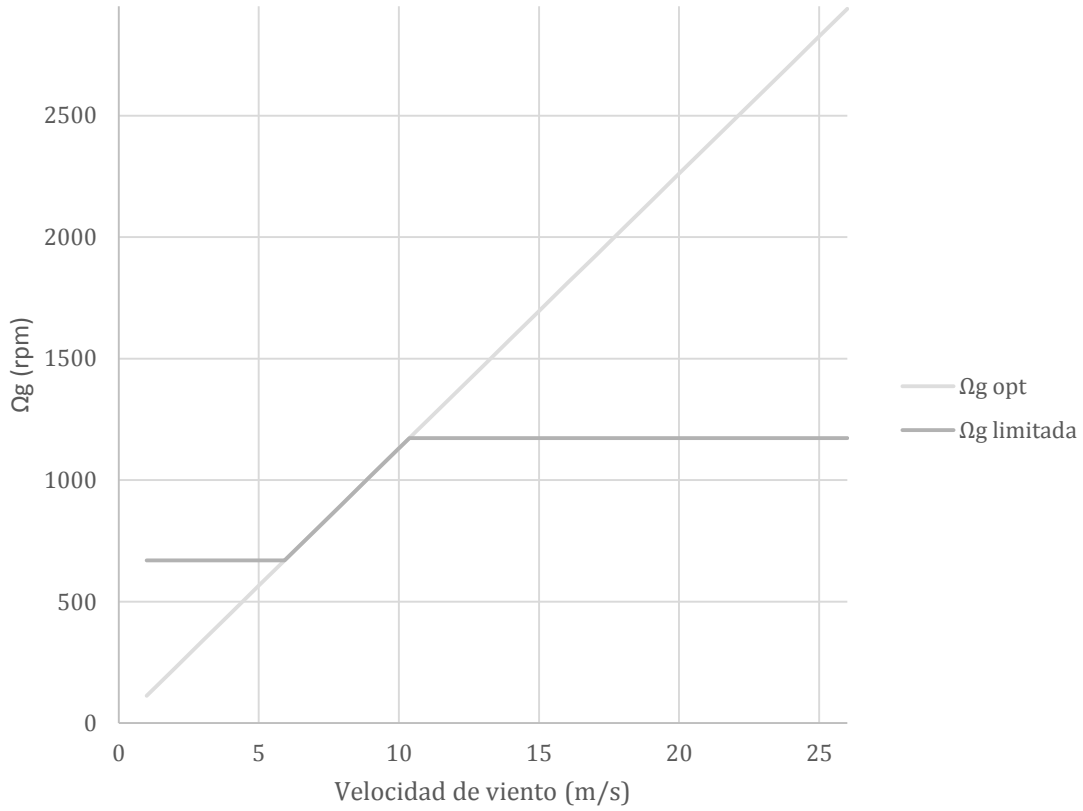

Figura 8. Velocidad de giro del generador VS velocidad de viento

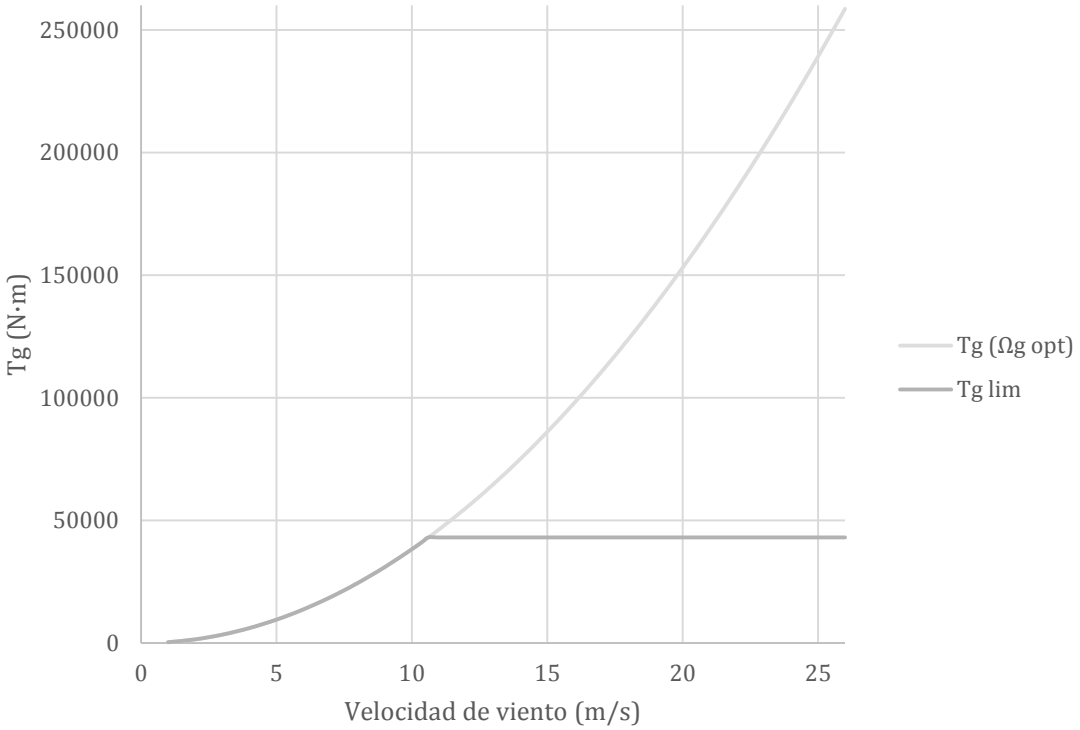

Figura 9. Par del generador VS velocidad de viento

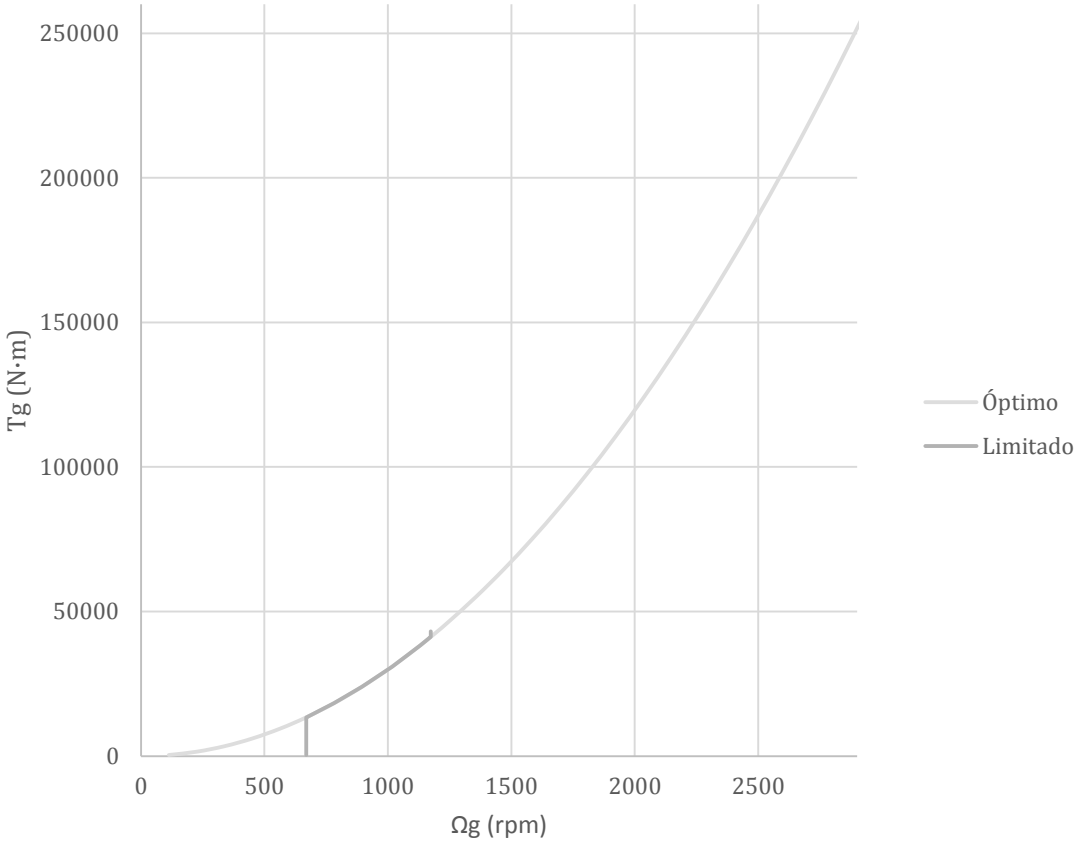

Figura 10. Par del generador VS velocidad de giro del generador

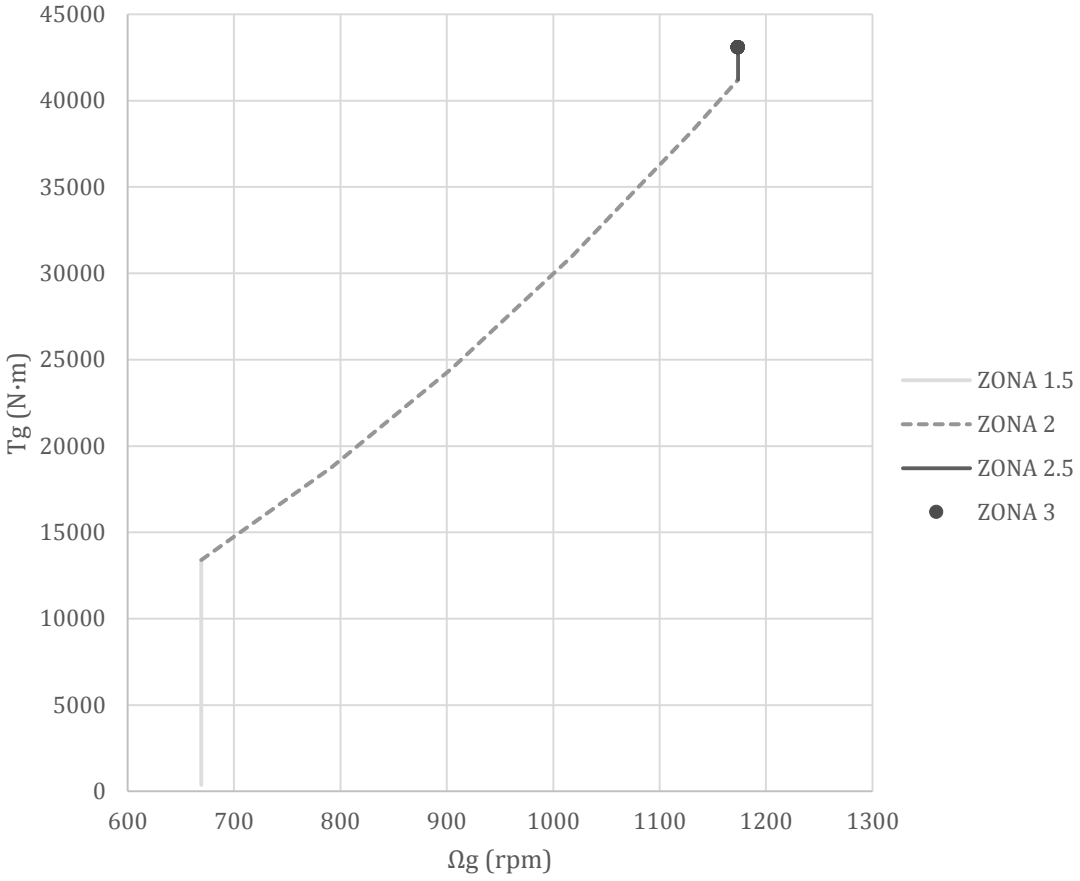

Figura 11. Zoom zonas. Par del generador VS velocidad de giro del generador

A partir de la Figura 11 se pueden identificar 4 zonas de operación cuyas características se recogen en la Tabla 1.

Zona		Condiciones			
		Ω_g	T_g	β	P
1.5	Primera vertical	$\Omega_{\mathrm{g}} = \Omega_{g,min}$	$T_g < K_{opt} * \Omega_{g,min}^2$	-	$P < P_{nom}$
2	Zona cuadrática	$\Omega_{g,\mathrm{min}} < \Omega_g < \Omega_{g,nom}$	$K_{opt} * \Omega_{g,min}^2 < T_g < K_{opt} * \Omega_{g,nom}^2$	-	$P < P_{nom}$
2.5	Segunda vertical	$\Omega_{\mathrm{g}} = \Omega_{g,nom}$	$K_{opt} * \Omega_{g,nom}^2 < T_g < T_{g,nom}$	-	$P < P_{nom}$
3	Zona nominal	$\Omega_{\mathrm{g}} = \Omega_{g,nom}$	$T_g = T_{g,nom}$	$\beta > 0$	$P = P_{nom}$

Tabla 1. Características de cada zona de operación del aerogenerador

En las primeras tres zonas (1.5, 2 y 2.5) el control del aerogenerador se realiza a través del control de par manteniendo el pitch constante. Sin embargo, en la zona 3 o zona nominal, el control del aerogenerador se realiza manteniendo el par constante a su valor nominal ($T_{g,nom}$) y manteniendo constante la velocidad de giro ($\Omega_{g,nom}$) a través del control del paso de pala o pitch (β).

El coeficiente de potencia sólo es máximo o próximo a su valor máximo (3) cuando el aerogenerador trabaja siguiendo la curva cuadrática. En la Figura 12 se puede observar que a

las velocidades de viento correspondientes a la zona 2, el Cp es próximo al máximo. Su valor no se corresponde exactamente con el máximo al haber tenido en cuenta el rendimiento del generador a la hora de obtener las gráficas (94.4%).

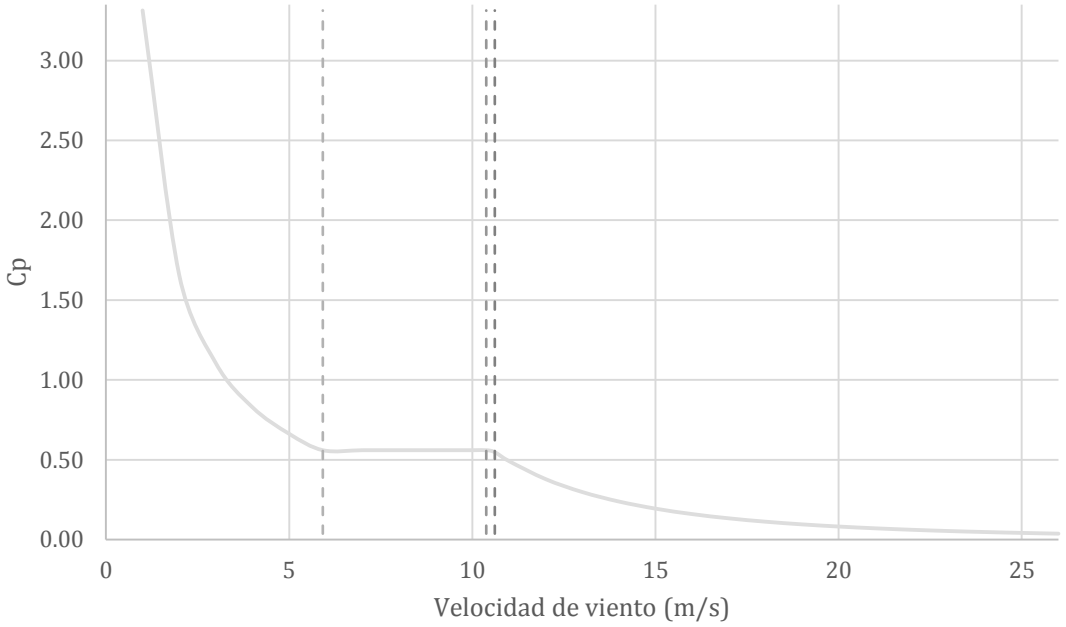

Figura 12. Cp VS velocidad de viento

Máquina de 5 MW de NREL

Las anteriores figuras relacionadas con las zonas de operación (Figura 7 a Figura 12) han sido obtenidas a partir de las características de la máquina de 5 MW de NREL (National Renewable Energy Laboratory). Dicho aerogenerador se sitúa a nivel internacional como referente de estudio al ser de libre acceso, ya que, por confidencialidad, las empresas dedicadas al diseño de aerogeneradores, no publican los datos de sus máquinas.

NREL es un laboratorio nacional de la Oficina de Eficiencia Energética y Energía Renovable del Departamento de Energía de Estados Unidos. Dentro de dicho laboratorio, el NWTC (National Wind Technology Center) es el encargado del estudio de la potencia del viento y del agua. Tanto la información del aerogenerador de 5 MW como los softwares empleados, se han obtenido del portal de información del NWTC [4].

Figura 13. Instalaciones del National Wind Technology Center en Colorado.
Fuente: https://www.nrel.gov/nwtc/

Las principales características de la máquina de 5 MW de NREL escogida vienen descritas en su manual [5]:

Potencia nominal	5 MW
Orientación del rotor, configuración	Barlovento, 3 palas
Control	Velocidad variable, Pitch colectivo
Tren de potencia	Eje de alta velocidad, multiplicadora
Diámetro del rotor	126m
Diámetro del buje	3 m
Altura del buje	90 m
Velocidad de viento: - de conexión - nominal - de corte	 3 m/s 11.4 m/s 25 m/s
Velocidad de rotor: - de conexión - nominal	 6.9 rpm 12.1 rpm
Masa del rotor	110000 kg
Masa de la góndola	240000 kg
Masa de la torre	347460 kg

Tabla 2. Características máquina 5 MW de NREL [5]

Además, a partir de su archivo primario .fst, que describe los parámetros de operación del aerogenerador y su geometría básica, se completa la siguiente información:

Par nominal en zona 3 (VS_RtTq)	43093.55
Condiciones ambientales. Gravedad	9.80665
Distancia eje del rotor a base de pala (HubRad)	1.5 m
Ángulo cónico de las palas (PreCone)	-2.5 deg
Ángulo azimuth	0 deg
Masa del buje (HubMass)	56800 kg
Inercia - Góndola (NacYIner) - Generador (GenIner) - Buje (HubIner)	 2607893 kg·m^2 534 kg·m^2 115925 kg·m^2
Rendimiento multiplicadora	100%
Rendimiento generador	94.4%
Ratio multiplicadora	97

Tabla 3. Características máquina 5 MW de NREL .fst [5]

Herramientas CAE

La ingeniería asistida por ordenador o CAE (*Computer Aided Engineering*) consiste en el empleo de programas informáticos para simular, analizar, validar y optimizar diseños de ingeniería, ya sean de productos o de procesos. El uso de dichos software permite reducir los costes y especialmente el tiempo.

Con el avance de los ordenadores, se han ido creando nuevas y mejores herramientas CAE. En el ámbito de la energía eólica se emplean CAE relacionadas con el diseño de aerogeneradores y con su integración en red. Debido a la previa suposición de considerar el generador y su esquema eléctrico como un bloque demandante de par (Figura 4), los CAE de integración en red no van a ser empleados ni estudiados.

Con el objetivo de trabajar con la máquina de 5 MW de NREL, se emplean varios software desarrollados por el NWTC, diferenciándolos en tres grupos principales: pre-procesadores, simuladores y post-procesadores. Su uso está estrechamente relacionado entre sí hasta formar una red de herramientas que permite crear modelos, simularlos y analizar sus resultados.

Principalmente se va a trabajar con uno de los simuladores, FAST (*Fatigue Aerodynamics Structures Turbulence*), al ser la herramienta de simulación referente y análoga a BLADED, software comercial creado por la entidad noruega DNV GL y el usado principalmente en la industria.

Pre-procesadores

Las herramientas de NREL categorizadas como pre-procesadores se emplean para crear los modelos tanto de los elementos del aerogenerador como del viento y emplearlos posteriormente en los simuladores.

- TurbSim. Herramienta que simula turbulencias estocásticas y genera campos de viento completo para emplear en AeroDyn e InflowWind.
- BModes. Código de elementos finitos que obtiene los modos dinámicos de las palas.
- IECWind. Software para crear archivos de viento IEC[4] a la altura del buje para emplear en los simuladores AeroDyn e InflowWind.
- Modes. Código de elementos finitos que crea los modos de las palas y la torre a partir de su geometría y estructura.
- AirfoilPrep. Documento de Excel que genera tablas de perfiles aerodinámicos para emplear en AeroDyn y WT_Perf.

Simuladores

Los simuladores se emplean principalmente para realizar simulaciones en dominio temporal. Dentro de este grupo de CAE se encuentra FAST, así como una larga lista de software que ofrece NWTC entre ellos:

[4] Comisión Electrotécnica Internacional (International Electrotechnical Commission): organización de normalización en los campos eléctrico, electrónico y tecnologías relacionadas. IEC 61400 corresponde con el estándar internacional relacionado con aerogeneradores [8].

- AeroDyn: rutinas de análisis aerodinámico para turbinas de eje horizontal. Se emplea para interactuar con paquetes de software de análisis dinámico (como FAST) para el análisis aeroelástico de los modelos de las turbinas.
- InflowWind: módulo para leer e interpolar archivos de viento.
- TurbSim: simulador primario estocástico de campo completo y turbulento para emplearlo con las herramientas de simulación InflowWind y AeroDyn.

FAST

Fatigue Aerodynamics Structures Turbulence (FAST) es una herramienta aeroelástica de ingeniería asistida por ordenador para simular la respuesta dinámica de aerogeneradores [6]. Tiene dos formas de operación o modos de análisis (AnalMode):

- Simulación o ejecución temporal de las ecuaciones no lineales de movimiento. Durante la simulación, la respuesta aerodinámica y estructural del aerogenerador ante unas condiciones de viento es determinada en el tiempo. Las salidas de la simulación incluyen series temporales de datos de las cargas aerodinámicas, así como cargas y deflexiones sobre la estructura. Dichas salidas pueden ser usadas para predecir cargas extremas y fatiga sobre la máquina.
- Linealización. FAST tiene la capacidad de extraer la representación lineal del sistema aeroelástico no lineal completo modelado en FAST. Esta capacidad de análisis es útil para obtener las matrices del espacio de estados del aerogenerador ('planta' del sistema) y contribuir al diseño y análisis de su control.

La versión de FAST con la que se va a trabajar es v7.02.00d-bjj del 20-Feb-2013. Los principales archivos empleados y su descripción se recogen en la siguiente tabla.

Nombre	Descripción
linearization.fst	Archivo principal de entrada de FAST. Incluye los parámetros de la máquina (configuración de la turbina, masa e inercia, eje de potencia, generador, plataforma, torre, góndola, freno, palas…), así como el modo de operación de FAST (simulación o linealización). También incluye el tiempo y paso de simulación, y la lista de los nombres de los parámetros de salida.
linearization.lin	Archivo de salida generado por FAST que incluye los resultados de la linealización, tanto los puntos de equilibrio como las matrices del espacio de estados.
DISCON.in	Contiene la información del control del aerogenerador. Actúa a modo de DLL e incluye variables como `CutInWindSpeed`, `CutOutWindSpeed` o `VS_MaxTq` (par máximo del generador en zona nominal)
aerodyn_gen.ipt	Contiene los parámetros aerodinámicos. Hace referencia tanto a los archivos con las propiedades aerodinámicas de

	pala (.dat) como al archivo de viento (.wnd). Relaciona los archivos aerodinámicos con los nudos de la pala empleados en el análisis.
Cylinder1.dat Cylinder2.dat DU21_A17.dat DU25_A17.dat DU30_A17.dat DU35_A17.dat DU40_A17.dat NACA64_A17.dat	Propiedades aerodinámicas distribuidas de pala. Cada .dat contiene la descripción completa del perfil de una sección de la pala (de -180º a +180º).
wind_det.wnd	Archivo que contiene la información del viento. Su extensión .wnd indica que se trata de un viento determinista, es decir, que el viento se conoce con certeza. Según con qué viento se quiera simular o linealizar con un viento, este archivo debe ser modificado. También podría emplearse un viento estocástico, es decir, que el viento no se conoce con anticipación, apareciendo el azar y la incertidumbre en la definición del viento. Se generan con Turbsim y se emplean los archivos .sum, .hh y .wnd.
Blade.dat	Contiene las características de la pala. Está organizado en 4 bloques: - Parámetros de la pala - Factores de ajuste - Propiedades distribuidas de la pala - Modos de la pala (fore-aft y side-to-side)
Tower.dat	Contiene las características de la torre. Está organizado en 5 bloques: - Parámetros de la torre - Factores de ajuste - Propiedades distribuidas de la torre - Modos fore-aft de la torre - Modos side-to-side de la torre
Param_linearization.dat	Archivo de control de linealización. Incluye el número de entradas y perturbaciones a la hora de linealizar, así como el tipo de ajuste (búsqueda del par, del pitch o de orientación-yaw).
FAST_SFunc.mexw32 FAST_SFunc-mexw64	Función que actúa de interfaz entre Simulink y FAST. Permite simular con Simulink el sistema de FAST integrándolo en la propia estructura de Simulink.

FAST_input_SFunc.fsm	Archivo de salida que contiene un resumen de la información generada por FAST al activar el flag SumPrint. Incluye información sobre las características de la turbina, el paso de simulación y un resumen de las propiedades del rotor y torre.
FAST_input_SFunc.opt	Archivo de salida que contiene un resumen de los datos geométricos y aerodinámicos de la pala generado por la combinación de FAST y AeroDyn al activar el flag SumPrint.
FAST_input_SFunc.outb FAST_input_SFunc.out	Archivos de salida (binario .outb y texto .out) que contienen en formato columnas la información temporal de las salidas indicadas en el archivo .fst al simular con FAST.

Tabla 4. Descripción de los archivos empleados con FAST

Post-procesadores

Las herramientas post-procesadoras son las encargadas de analizar los resultados de las simulaciones. Los principales softwares creados por NWTC y basados en scripts de MATLAB son:

- MExtremes. Genera tablas de eventos extremos para una o varias simulaciones. Esta función también está disponible con MCrunch, pero al emplear una sola serie temporal, su ejecución es más rápida y es más notable cuantas más simulaciones se quieran analizar.
- MLife. Herramienta creada para calcular la fatiga y vida de una o varias simulaciones. Podría emplearse MCrunch con el mismo objetivo de análisis de vida, pero al trabajar con una sola serie temporal, emplea menos memoria y es más rápido a la hora de procesar muchos archivos.
- MCrunch. Es un conjunto de scripts de MatLab para el análisis de los datos de las simulaciones. Permite obtener gráficas tanto de eventos extremos como de fatiga y vida, También permite obtener histogramas con la densidad de probabilidad de la salida de la simulación seleccionada. MCrunch es una herramienta muy completa con mayores posibilidades que MExtremes y MLife pero con el gran inconveniente que al trabajar con todas las salidas de las simulaciones, al comparar varios archivos, su ejecución es más lenta.

Modelo lineal con FAST

La mayoría de sistemas reales son no lineales. Según sus características, las ecuaciones no lineales que conforman el modelo matemático pueden ser de gran complejidad. Por ello, el uso de linealizaciones es de gran utilidad, tanto para el estudio de las propiedades del sistema como para la posterior etapa de diseño.

FAST tiene la capacidad de extraer el modelo lineal a partir del modelo aeroelástico no lineal de un aerogenerador en un punto de operación. Para ejecutar FAST en modo linealización, en el archivo principal (.fst), se debe introducir el valor 2 en el parámetro AnalMode (Figura 14).

```
---------------------- SIMULATION CONTROL --------------------------------------
False       Echo        - Echo input data to "echo.out" (flag)
   1        ADAMSPrep   - ADAMS preprocessor mode {1: Run FAST, 2: use FAST as a preprocessor to create an ADAMS model, 3: do both}
   2        AnalMode    - Analysis mode {1: Run a time-marching simulation, 2: create a periodic linearized model} (switch)
   3        NumBl       - Number of blades (-)
   600      TMax        - Total run time (s)
 0.0200     DT          - Integration time step (s)
```

Figura 14. AnalMode 2

Con el fin de homogeneizar el trabajo, se emplea una misma máquina para la linealización y el cierre de lazos. Esto hace que no sea necesario modificar los parámetros del aerogenerador y los archivos relacionados con la aerodinámica de los perfiles de la pala y de la torre.

Las últimas líneas de código del archivo principal de FAST (linearization_gen.fst) se emplean para seleccionar los parámetros de salida de la linealización. Con el objetivo de homogeneizar y estandarizar el trabajo, se emplean las salidas de la Figura 15 tanto para la linealización como para la simulación.

```
"RotSpeed"                      - Rotor velocity (rpm)
"GenSpeed"                      - Generator velocity (rpm)
"GenTq"                         - Generator torque (kN·m)
"BlPitch1"                      - Blade pitch angle (deg)
"TipSpdRat"                     - Tip Speed Ratio
"NcIMUTVxs"                     - Nacelle-x-velocity (m/s)
"NcIMUTAxs"                     - Nacelle-x-acceleration (m^2/s)
"RootMyb1"                      - Blade 1 flapwise moment My (kN·m)
"RootMxb1"                      - Blade 1 edgewise moment Mx (kN·m)
"LSShftMxa"                     - LSS torque (EQUIVALENT TO THE ROTOR TORQUE)
"TwrBsMyt"                      - Tower base pitching torque (kN·m)
"GenPwr"                        - Generator Power (kW)
```

Figura 15. Parámetros de salida seleccionados

Se seleccionan dichos parámetros y en ese determinado orden por similitud a la estandarización empleada en el sector comercial de la energía eólica. Además, contiene los parámetros más importantes empleados a la hora de diseñar y de controlar el sistema. La mayor diferencia con softwares comerciales de simulación de aerogeneradores son las unidades con las que trabaja FAST, ya que no se corresponden con las unidades del Sistema Internacional.

Linealización

A la hora de linealizar con FAST, el archivo **Param_linealization.dat** se encarga de definir los parámetros de la linealización. En el apartado de entradas y perturbaciones, se definen el número y las variables de entrada de control del sistema. También se definen el número de perturbaciones de viento y el tipo. Las principales variables de entradas y perturbaciones se recogen en la Tabla 5.

Variable	Valor
NInputs	0 a 4+(Número de Palas)
CntrlInpt	1: ángulo de orientación de la góndola 2: velocidad de orientación de la góndola 3: par del generador 4: ángulo colectivo de paso de pala 5: ángulo individual de paso de pala 1 6: ángulo individual de paso de pala 2 7: ángulo individual de paso de pala 3
NDisturbs	0 a 7
Disturbnc	1: velocidad de viento horizontal a la altura del buje 2: dirección de viento horizontal 3: velocidad de viento vertical 4: cizalladura horizontal del viento 5: rugosidad vertical de cizalladura 6: cizalladura lineal vertical del viento 7: ráfaga de viento horizontal a la altura del buje

Tabla 5. Variables de entradas y perturbaciones

Para el caso que se va a estudiar, se consideran entradas del sistema el par del generador (Tcntrl) y el ángulo colectivo de paso de pala o pitch (Cpitch). Como perturbaciones de viento, se va a linealizar suponiendo únicamente la perturbación de velocidad de viento horizontal a la altura de buje u *'horizontal hub-height wind speed' (*Figura 16*)*.

```
---------------------- INPUTS AND DISTURBANCES ------------------------------
   2        NInputs     - Number of control inputs [0 (none) or 1 to 4+NumBl] (-)
   3,4      CntrlInpt   - List   of control inputs [1 to NInputs] {1: nacelle yaw
   1        NDisturbs   - Number of wind disturbances [0 (none) or 1 to 7] (-)
   1        Disturbnc   - List   of input wind disturbances [1 to NDisturbs] {1: h
```

Figura 16. Entradas y perturbaciones escogidas

Para poder crear una solución periódica en estado estacionario, en el archivo de entrada Param_linealization.dat hay que definir el tipo de ajuste que debe realizar FAST (TrimCase, Figura 17):

- Búsqueda de la orientación de la góndola (1)
- Búsqueda del par generador (2)
- Búsqueda del ángulo colectivo de paso de pala (3)

En el caso de no tener que calcular las condiciones periódicas en estado estacionario, es decir, que FAST linealice en torno a las condiciones iniciales, la variable CalcStdy deberá tener valor '*False*'.

```
---------------------- PERIODIC STEADY STATE SOLUTION -----------------------
True       CalcStdy    - Calculate periodic steady state condition {False: linear
      3    TrimCase    - Trim case {1: find nacelle yaw, 2: find generator torque,
   0.01    DispTol     - Convergence tolerance for the 2-norm of displacements in
   0.02    VelTol      - Convergence tolerance for the 2-norm of velocities    in
```

Figura 17. Ajuste de la solución periódica en estado estacionario

Tal y como se desarrolla en el apartado Zonas de operación, en las zonas 1.5, 2 y 2.5 el pitch es constante mientras que el control del aerogenerador se realiza por medio del control de par. Sin

embargo, en la zona 3, el par se fija a su valor nominal mientras que se realiza el control de pitch. Por lo tanto, en función de la velocidad de viento y, por lo tanto, de la zona de operación del aerogenerador, la variable TrimCase tendrá valor 2 o 3.

Además de Param_linealization.dat, para realizar correctamente la linealización, el archivo principal de FAST (**linearization.fst**) debe incluir un valor estimado del pitch inicial en grados (Figura 18) y de la velocidad de giro del rotor en rpm (Figura 19).

```
46    23.8      BlPitch(1)   - Blade 1 initial pitch (degrees)
47    23.8      BlPitch(2)   - Blade 2 initial pitch (degrees)
48    23.8      BlPitch(3)   - Blade 3 initial pitch (degrees)
```

Figura 18. Ejemplo valor pitch inicial

```
73    12.10      RotSpeed     - Initial or fixed rotor speed (rpm)
```

Figura 19. Ejemplo valor velocidad de giro del rotor inicial

FAST permite la obtención del modelo lineal de la máquina. Sin embargo, cada ejecución de FAST como herramienta de linealización, calcula los valores del espacio de estados así como el punto de equilibrio para el viento referenciado en **aerodyn_gen.ipt** (Figura 20).

```
10  "../../wind/deterministic/wind_det.wnd"       WindFile - Name of file containing wind data
```

Figura 20. Viento aerodyn_gen.ipt

El archivo de viento .wnd que contiene los datos del viento debe corresponderse con un viento constante a la velocidad de viento entorno a la que se quiera linealizar (Figura 21).

```
14    ! Time Wind    Wind    Vertical    Horiz.  Pwr.Law Lin.Vert.   Gust
15    !  Speed   Dir Speed   Shear   Vert.Shr    Shear   Speed
16    ! (sec)    (m/s)    (Deg)    (m/s)              (m/s)
17   0    19.0000 0   3.4793  0   0.2 0   0
18   600 19.0000 0   3.4793  0   0.2 0   0
```

Figura 21. Ejemplo viento linealización

La linealización se realiza sólo a la velocidad de viento correspondiente al archivo **wind_det.wnd**. Por ello, para obtener el modelo lineal completo de la turbina a diferentes velocidades de viento, el procedimiento de linealización se debe repetir para cada velocidad de viento.

El coste computacional de FAST no es excesivamente alto, por lo que se ha decidido realizar la linealización para todas las velocidades de viento de 3 m/s a 25 m/s con saltos de 1 m/s. Para ello se crea un **protocolo de identificación de zonas** con el propósito de definir correctamente las condiciones iniciales y el TrimCase para cada velocidad de viento.

Identificación de zonas

Para cada velocidad de viento, el punto de equilibrio es diferente. A pesar de que FAST computa dicho punto de operación al realizar la linealización, en su archivo principal **linearization.fst** se debe introducir una estimación del pitch inicial y de la velocidad de giro del rotor inicial.

Siendo V la velocidad de viento entorno a la que se va a linealizar, se realiza el cálculo de la velocidad de giro óptima del rotor (25). A partir de dicha velocidad se distinguen las zonas 1.5, 2 y 2.5 o 3, y se obtiene la velocidad de giro del rotor inicial.

$$\Omega_{r_{calc}} = V * \frac{\lambda_{\text{opt}}}{R} \tag{25}$$

La distinción entre la zona 2.5 y 3 se realiza por medio del par obtenido en la linealización con FAST. Si dicho par es superior al nominal, la linealización se ha realizado para la búsqueda de par (zona 2.5 – TrimCase 2) cuando debería haberse llevado a cabo para la búsqueda de pitch (zona 3 – TrimCase 3).

El pitch inicial para las zonas 1.5, 2 y 2.5 es elegido por el usuario (β_0), generalmente dándole un valor de 0 o inferior. Sin embargo, la obtención del pitch inicial en la zona 3 se realiza mediante la interpolación de la curva $Cp - \lambda - \beta$ (Figura 22) que caracteriza la máquina con los valores calculados λ_{calc} (26) y Cp_{calc} (27).

$$\lambda_{calc} = \Omega_{r,nom} * \frac{R}{V} \tag{26}$$

$$Cp_{calc} = \frac{2 * Gbx * T_{g,nom} * \Omega_{r,nom}}{\rho * \pi * R^2 * V^3} \tag{27}$$

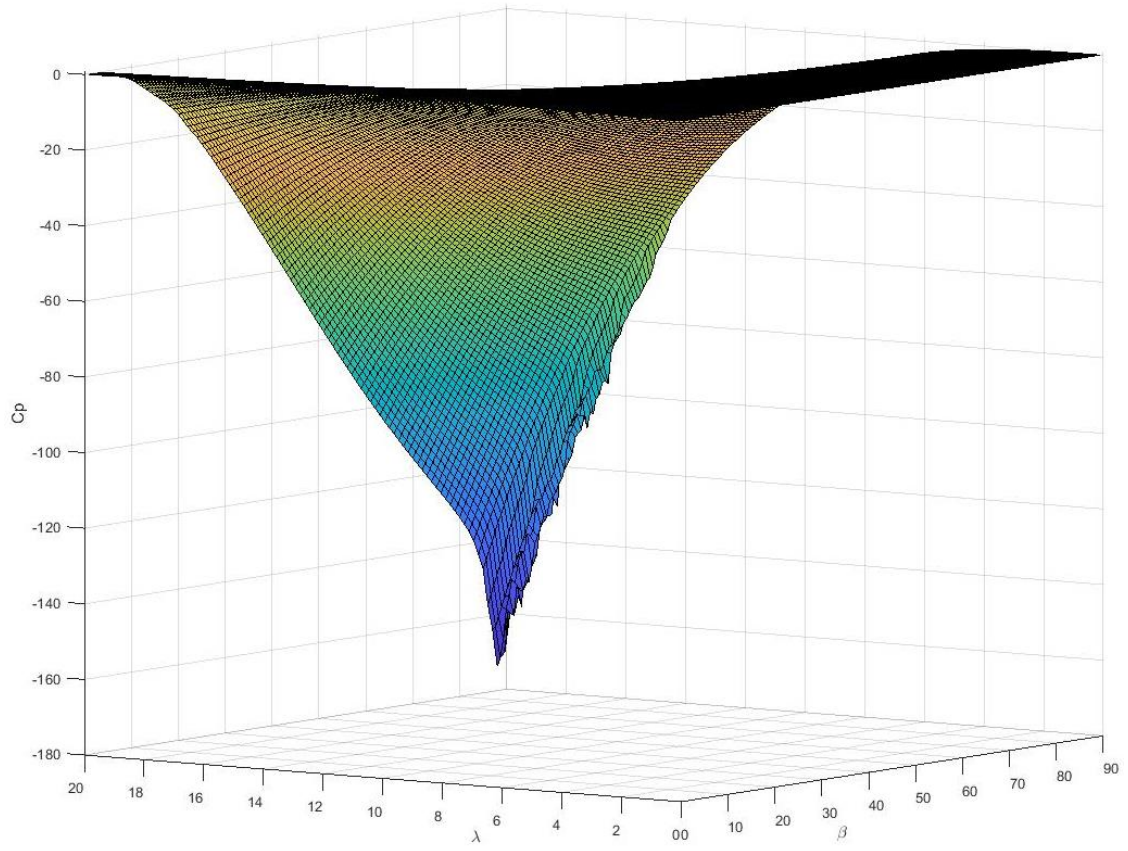

Figura 22. Curva $Cp - \lambda - \beta$ de la máquina de 5 MW de NREL

La siguiente tabla (Tabla 6) resume las condiciones que debe cumplir la linealización obtenida según la zona supuesta ($\Omega_{r,calc}, T_{g,lin}$), así como las condiciones iniciales y el TrimCase que deberían haberse impuesto en la simulación. En caso de no cumplir las condiciones $\Omega_{r,calc}$ y $T_{g,lin}$, deberá realizarse una nueva linealización para dicha velocidad de viento pero en diferente zona de operación y, por lo tanto, con diferentes condiciones iniciales y TrimCase.

$\Omega_{r_{calc}}$	$T_{g,lin}$	Zona	$\Omega_{r_{init}}$	β_{init}	TrimCase
$\Omega_{r_{calc}} < \Omega_{r.min}$	$T_{g,lin} < T_{g,nom}$	1.5	$\Omega_{r,min}$	β_0	2
$\Omega_{r.min} < \Omega_{r_{calc}} < \Omega_{r.nom}$		2	$\Omega_{r_{calc}}$	β_0	2
$\Omega_{r,nom} < \Omega_{r_{calc}}$		2.5	$\Omega_{r,nom}$	β_0	2
	$T_{g,lin} \geq T_{g,nom}$	3	$\Omega_{r,nom}$	β_{calc}	3

Tabla 6. Identificación de zonas en la linealización de FAST

Modelo lineal obtenido

FAST genera el archivo **linearization_gen.lin** con el modelo lineal de la velocidad de viento escogida en forma de texto. Para poder trabajar con la información de la linealización, se ha desarrollado la función GetSysFastSysturb basada en el script de GetMats de J. Jonkman de NREL. Dicha función es la encargada de leer el texto del archivo linearization_gen.lin y crear la estructura de la Figura 23. Estructura obtenida con GetSysFastSysturb.

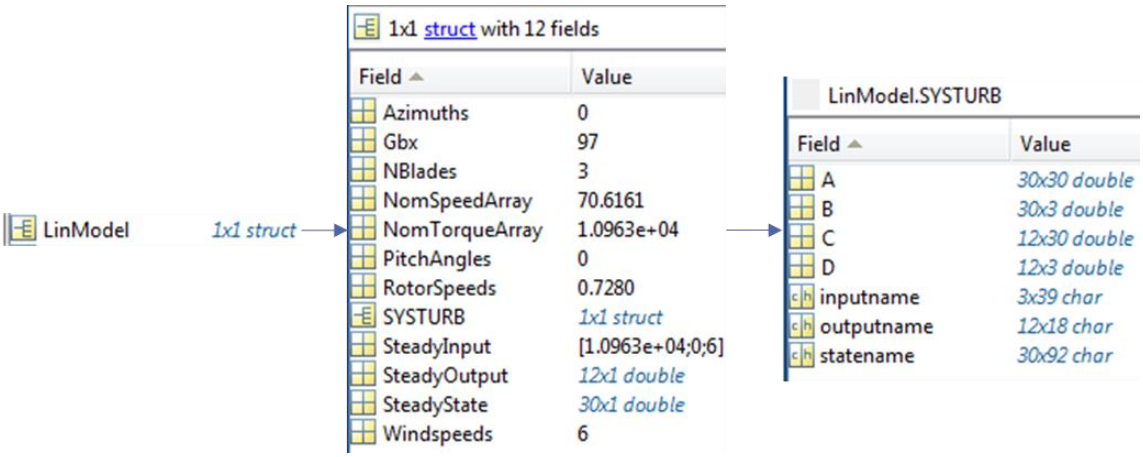

Figura 23. Estructura obtenida con GetSysFastSysturb

Se ha escogido la estructura LinModel con el objetivo de estandarizar la estructura de los modelos lineales. BLADED como software comercial de referencia emplea una estructura similar que incluye los puntos de equilibrio encontrados, así como las matrices del espacio de estados en una estructura llamada SYSTURB.

Los datos obtenidos en la Figura 23 corresponden a la lectura del archivo linearization_gen.lin para una velocidad de viento de 6 m/s (Windspeeds). Sin embargo, para poder obtener el modelo lineal para todas las velocidades de viento anteriormente propuestas (3 – 25 m/s), se deben realizar tantas linealizaciones como velocidades de viento.

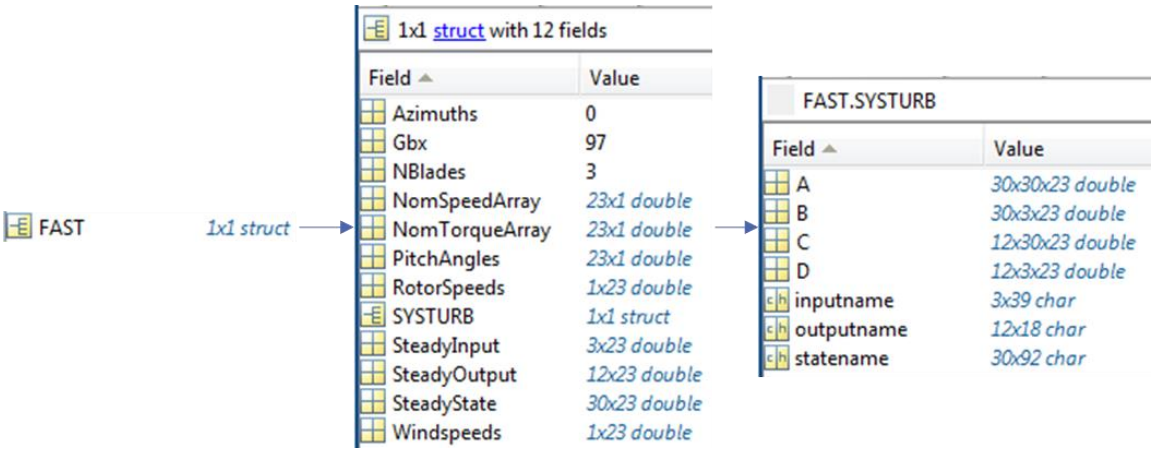

Figura 24. Estructura obtenida para todo el rango de vientos

En la Figura 24 se tiene la estructura completa obtenida tras realizar la linealización para cada una de las velocidades de viento. Los valores almacenados se corresponden con las matrices del espacio de estados (SYSTURB) y con los puntos de equilibrio encontrados para cada velocidad de viento. Las unidades de cada una de las variables de los puntos de equilibrio corresponden con las obtenidas directamente con FAST y están recogidas en la Tabla 7.

Variable		Unidades	Descripción
Azimuths		deg	Ángulo azimuth
Gbx		-	Relación de la multiplicadora
NBlades		-	Número de palas
NomSpeedArray		rad/s	Velocidad de giro del generador
NomTorqueArray		N·m	Par del generador
PitchAngles		rad	Ángulo de pitch
RotorSpeeds		rad/s	Velocidad de giro del rotor
SteadyInput	Electrical generator torque Rotor collective blade pitch Horizontal hub-height wind speed	N·m rad m/s	Variables de entrada: Par del generador Ángulo de pitch Velocidad de viento
SteadyOutput	RotSpeed GenSpeed GenTq BlPitch1 TipSpdRat NcIMUTVxs NcIMUTAxs RootMyb1 RootMxb1 LSShftMxa TwrBsMyt GenPwr	rpm rpm kN·m deg - m/s m/s^2 kN·m kN·m kN·m kN·m kW	Variables de salida: Velocidad de giro del rotor Velocidad de giro del generador Par del generador Ángulo de pitch de la pala 1 λ Velocidad de la góndola en el eje X Aceleración de la góndola en el eje X Momento My flapwise de base de pala 1 Momento Mx edgewise de base de pala 1 Par del rotor Par base de torre Potencia generador
Windspeeds		m/s	Velocidad del viento

Tabla 7. Unidades y descripción salidas de la linealización con FAST

A partir de los datos obtenidos para cada punto de equilibrio, se construyen las gráficas de las figuras Figura 25, Figura 26 y Figura 27.

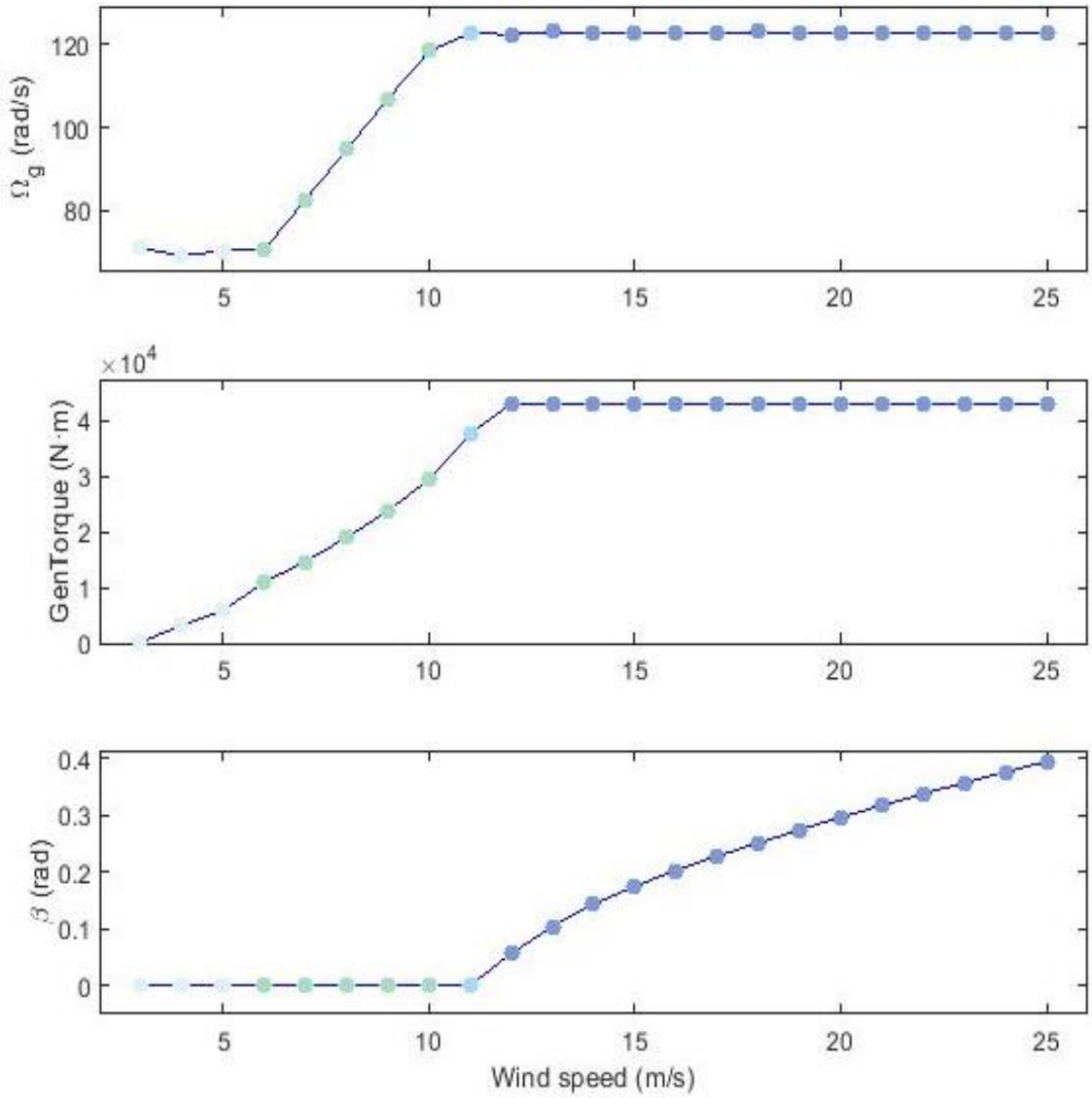

Figura 25. Ω_g, Tg y β para cada velocidad de viento

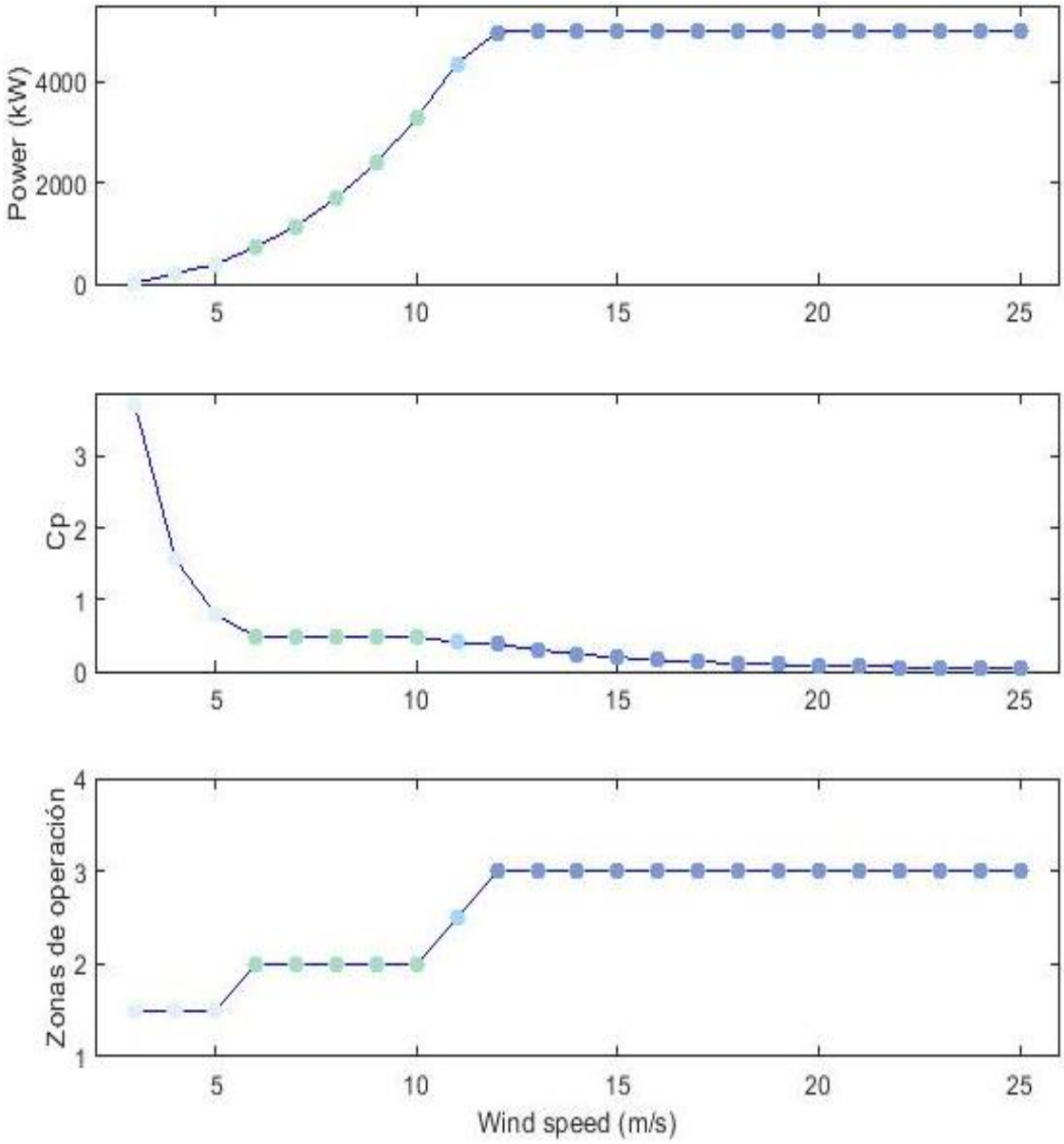

Figura 26. Potencia, Cp y zonas de operación para cada velocidad de viento

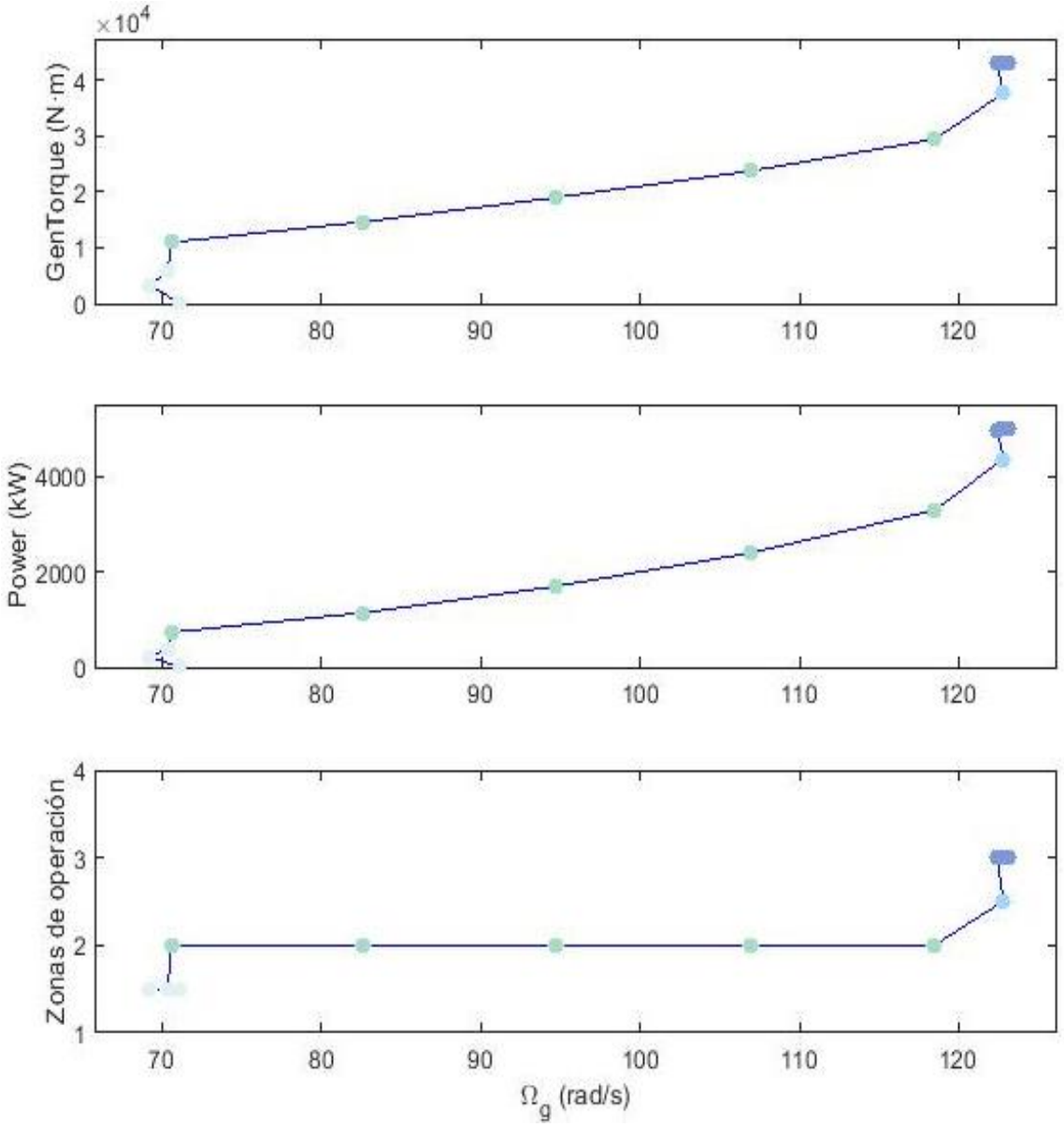

Figura 27. Tg, potencia y zonas de operación para cada Ω_g

Los resultados obtenidos en cuanto a los puntos de equilibrio se asemejan a los esperados teóricamente y desarrollados en el apartado Zonas de operación. Además de información sobre dichos puntos, a partir de las matrices del espacio de estados obtenidos en la linealización, se pueden construir los diagramas de Bode del sistema.

Las siguientes figuras (Figura 28, Figura 29, Figura 30, Figura 31, Figura 32, Figura 33) representan el diagrama de Bode para las velocidades de viento 4, 7, 11, 15 y 19 m/s para cada una de las entradas (pitch, par y viento) y las salidas Ω_g (rpm) y Potencia (kW). Los diagramas de Bode más empleados son los relacionados con la velocidad de giro del generador, ya que a partir de ellos se diseñan los principales controladores.

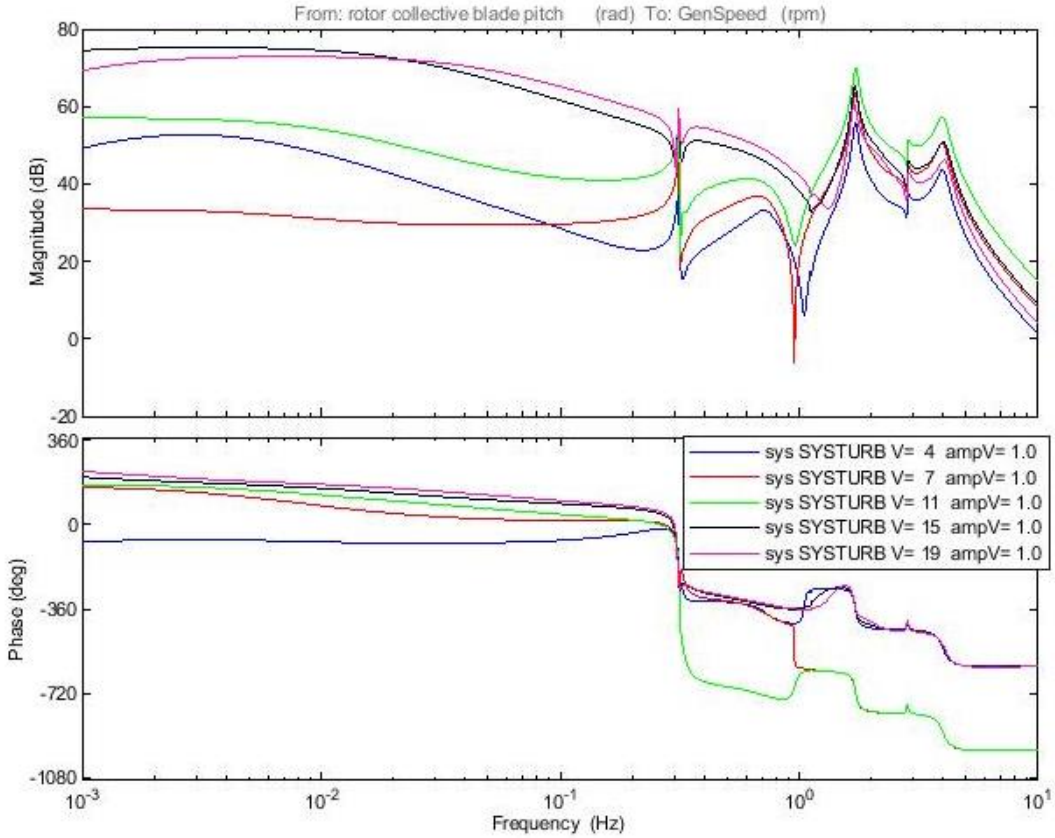

Figura 28. Bode de β (rad) a Ω_g(rpm)

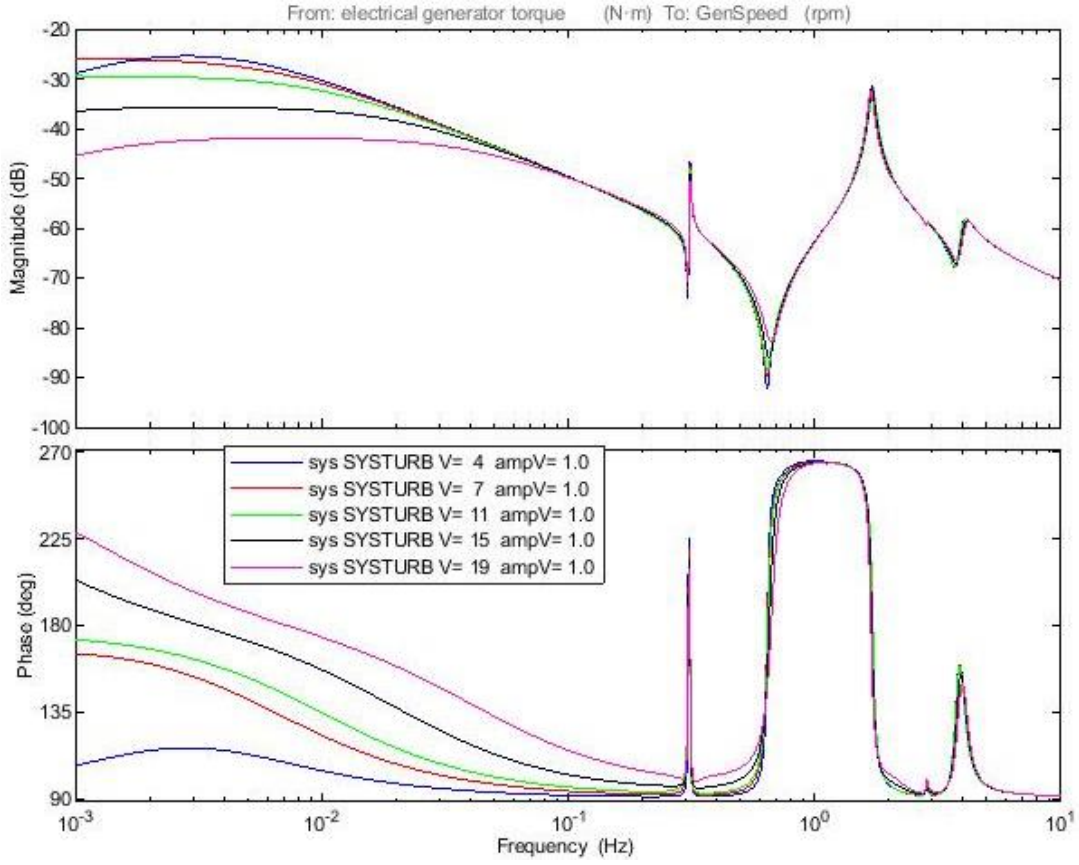

Figura 29. Bode de Tg (N·m) a Ω_g(rpm)

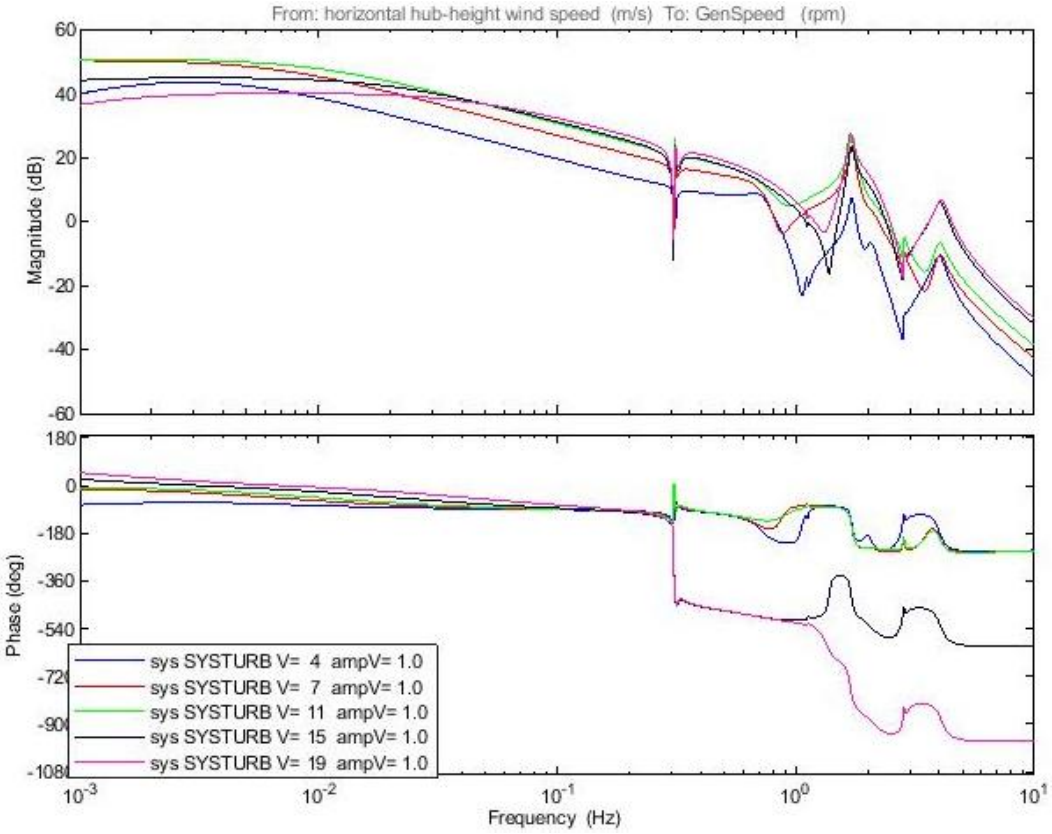

Figura 30. Bode de V (m/s) a Ω_g(rpm)

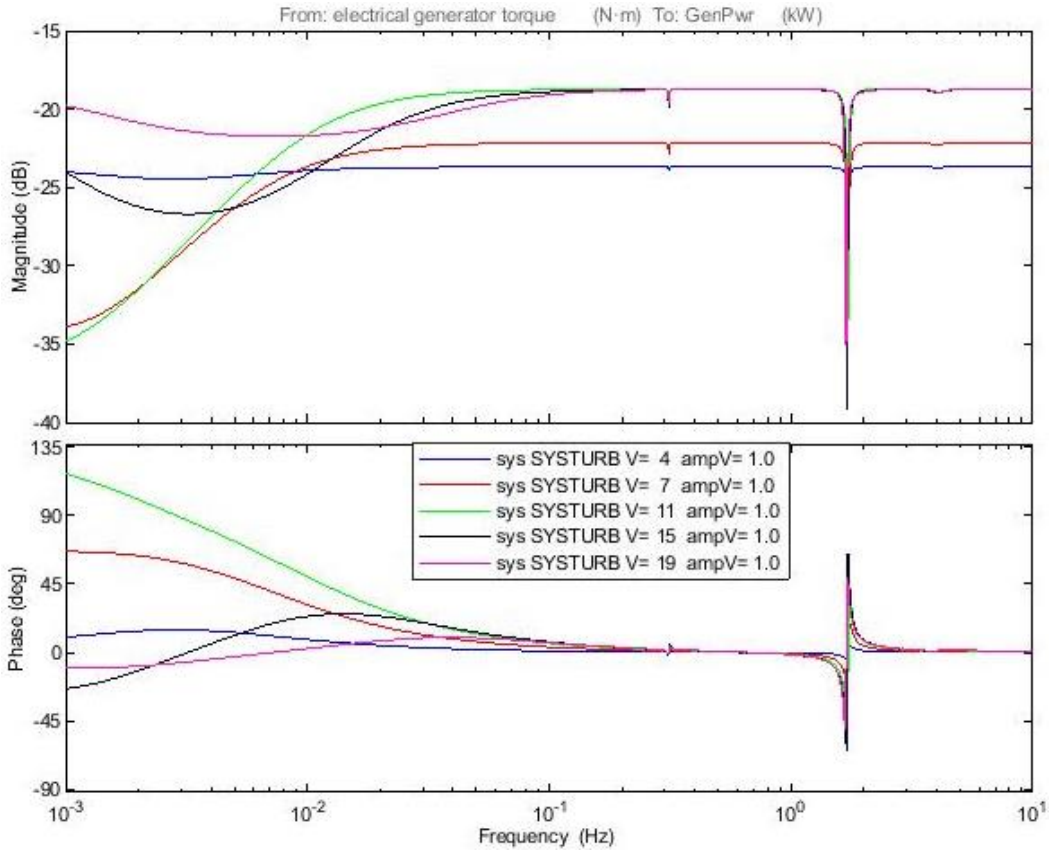

Figura 31. Bode de Tg (N·m) a Potencia (kW)

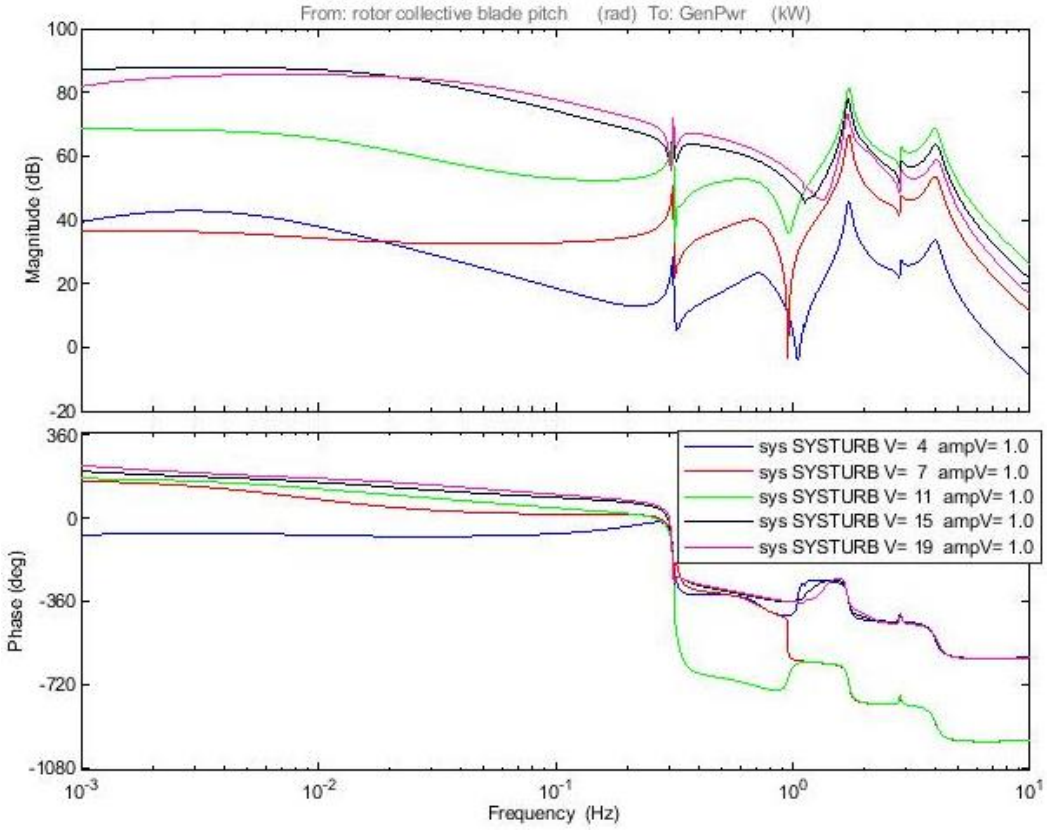

Figura 32. Bode de β (rad) a Potencia (kW)

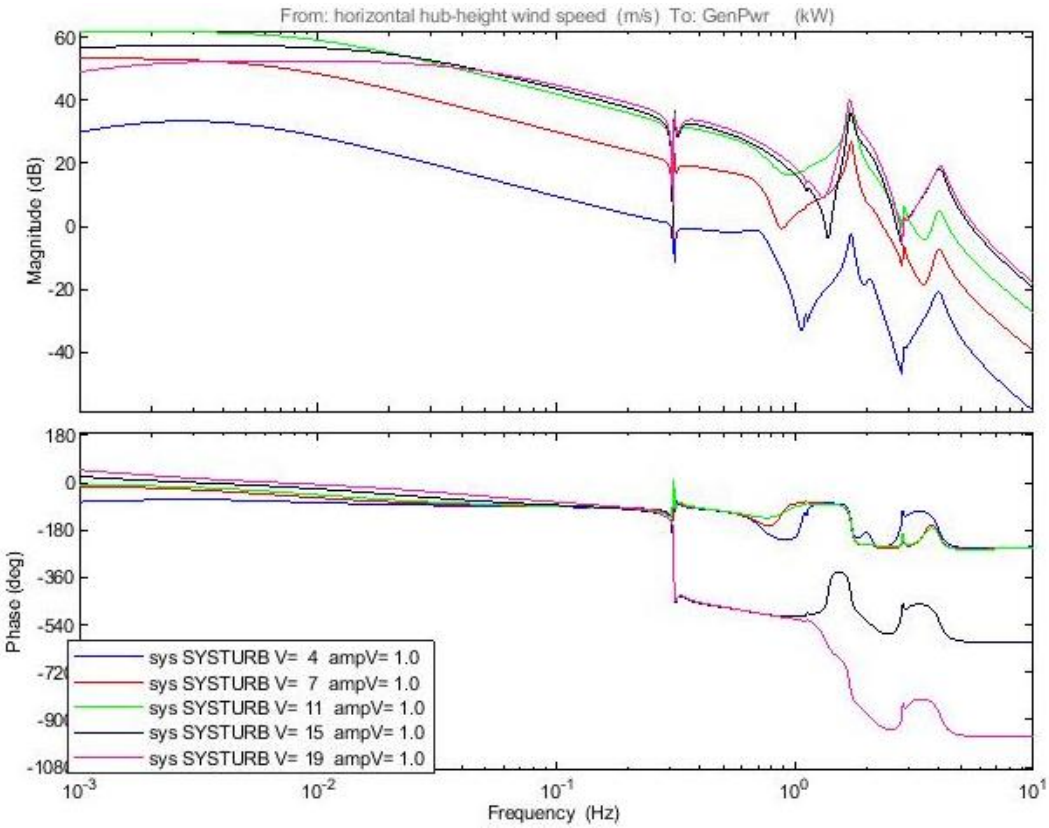

Figura 33. Bode de V (m/s) a Potencia (kW)

Tal y como se pueden observar en las anteriores figuras, el comportamiento del sistema varía fuertemente con la velocidad de viento. Esto implica que a la hora de diseñar el control del

sistema, no puede emplearse un solo controlador para todas las velocidades de viento, sino que el control debe distinguir no sólo entre zonas sino también entre velocidades de viento.

Con la estructura SYSTURB pueden obtenerse tantos diagramas de Bode como posibles relaciones entrada-salida se tienen del sistema. En el caso de obtener el diagrama de Bode para la salida Ω_r el resultado es el mismo que con $\Omega_g$pero con una diferencia constante de ganancia dada por el ratio de la multiplicadora (28) (Figura 34).

$$\Delta dB = 20 * \log(Gbx) = 39.735 \tag{28}$$

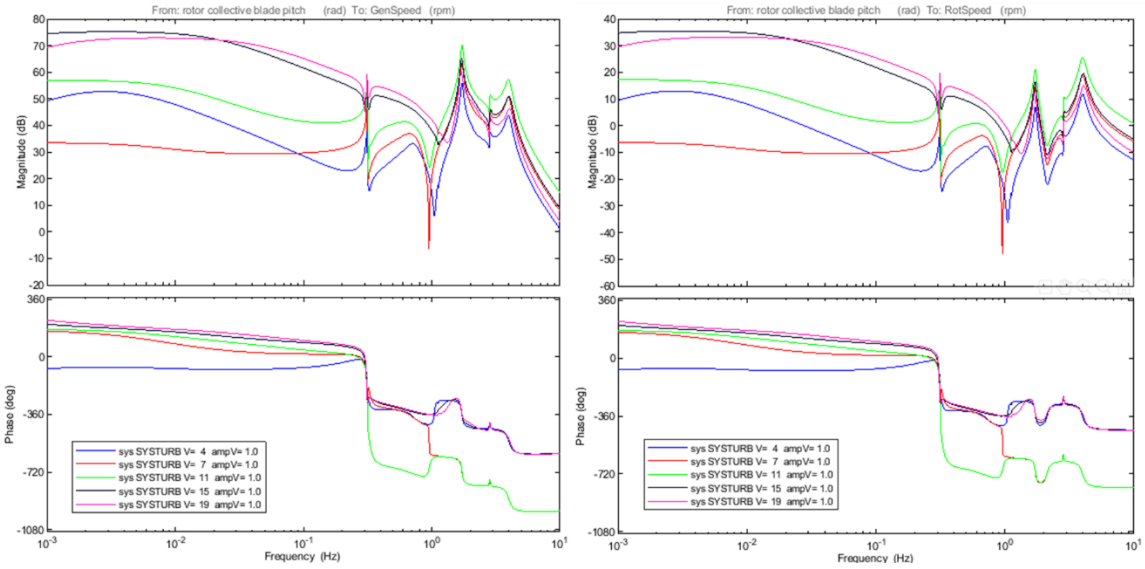

Figura 34. Desfase entre GenSpeed y RotSpeed

Al obtener los diagramas de Bode para la salida de par del generador (GenTg), los diagramas de entrada viento y pitch están vacíos, ya que una variación de la velocidad de viento o de pitch no implica una diferencia de par. Sin embargo, el diagrama de Bode correspondiente a la entrada de demanda de par y salida de par "real" tiene valor constante y diferente a 0 debido al cambio de unidades entre entrada y salida de par que hace FAST (29)(Figura 35).

En el caso de considerar el actuador de par como un sistema de primer o segundo orden, el comportamiento resultante sería diferente al obtenido. Sin embargo, no se ha introducido ninguna modificación en los archivos de FAST, por lo que se considera que el actuador de par es ideal. Esto implica que al demandarle a la máquina un par en N·m, esta responde produciendo ese mismo par aunque medido en kN·m.

$$\Delta dB = 20 * \log\left(\frac{N \cdot m}{kN \cdot m}\right) = -60 \tag{29}$$

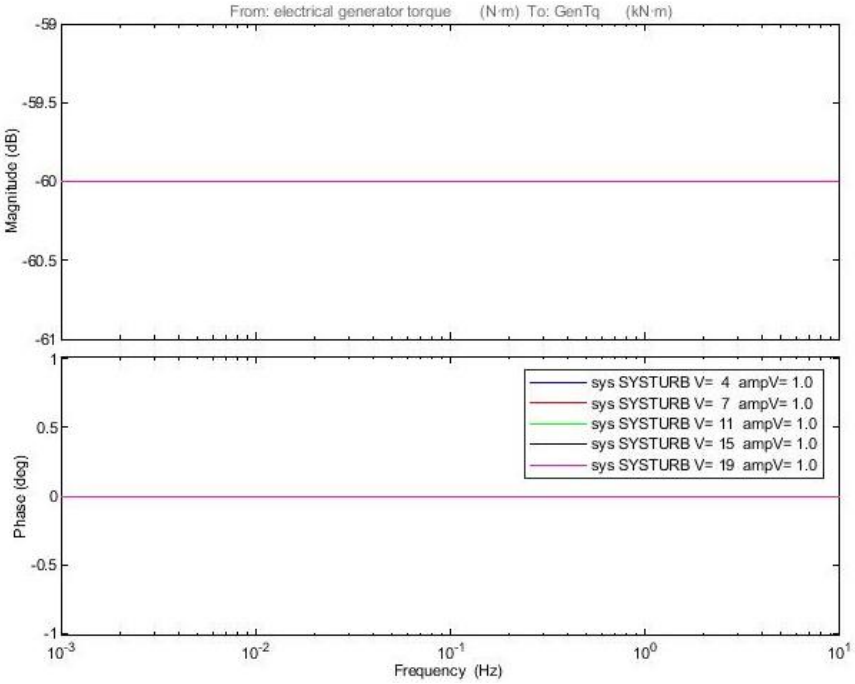

Figura 35. Bode de Tg (N·m) a Tg (kN·m)

De manera similar a como ocurre con el par, la relación entre demanda de ángulo de paso de pala (rad) y salida del actuador de pitch (deg) tiene un valor constante y diferente a 0 (30) (Figura 36), mientras que para la entrada de viento y de par, el diagrama de Bode está vacío.

Tal y como ocurría con el actuador de par, se ha supuesto que el actuador de pitch es ideal y, por lo tanto, sólo hay cambio de unidades. En el caso de suponer un actuador de par de primer orden, el diagrama de Bode obtenido cambiaría.

$$\Delta dB = 20 * \log\left(\frac{rad}{deg}\right) = 35.162 \qquad (30)$$

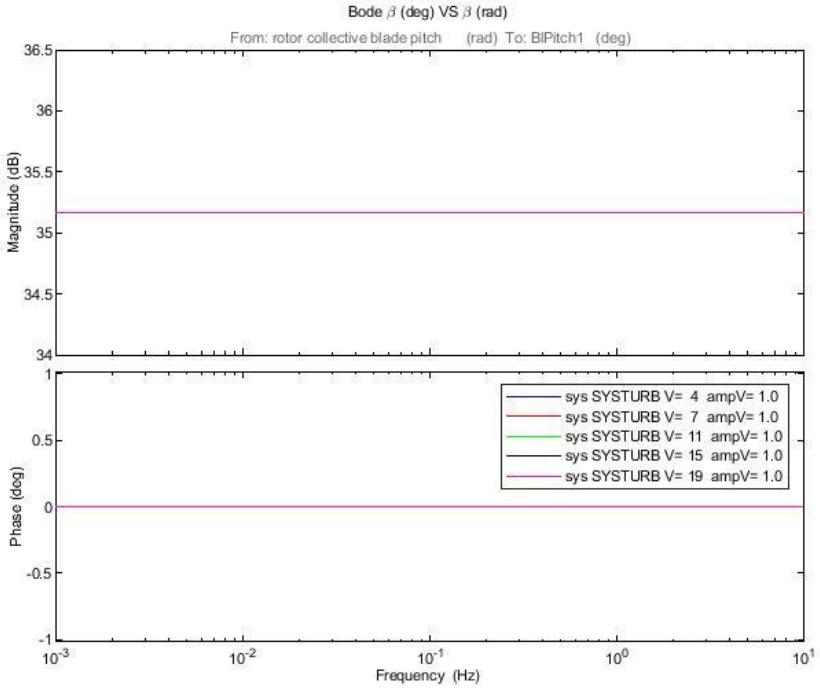

Figura 36. Bode de β (rad) a β (deg)

Estandarización

Al comienzo del apartado *Modelo lineal obtenido*, se decide seguir una estructura para el almacenamiento y operación de los datos de las linealizaciones. Hasta ahora, los proyectos basados en FAST, no han prestado especial atención a la estructura del modelo lineal. Sin embargo, la fuerte variabilidad de la presentación de los datos, conlleva incompatibilidades posteriores entre datos de diferentes proyectos.

Para evitar futuras incompatibilidades con post-procesadores, en este proyecto se pretende seguir la estructura que emplea la industria del sector eólico basada en el software comercial BLADED. Para ello, se estudia la estructura y características de los modelos lineales obtenidos con BLADED (Figura 37).

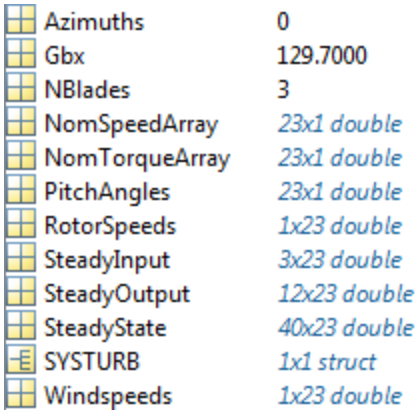

Figura 37. Ejemplo de estructura de BLADED

Al linealizar con BLADED, la información de salida incluye los puntos de equilibrio, así como las matrices del espacio de estados del modelo lineal. Todos ellos son calculados para cada velocidad de viento del vector fila Windspeeds (Figura 37, Figura 38). Dentro de la estructura también se incluye el número de palas, el azimuth y la relación de la multiplicadora (NBlades, Gbx y Azimuths respectivamente).

```
Windspeeds =

     3     4     5     6     7     8     9    10    11    12    13    14    15    16    17    18    19    20    21    22    23    24    25
```

Figura 38. Ejemplo Windspeeds de BLADED

Los puntos de equilibrio se recogen en las matrices SteadyInput, SteadyOutput y SteadyState según correspondan a las variables de entrada, las variables de salida o los estados del espacio de estados. Además, los valores de los puntos de equilibrio correspondientes a la velocidad del generador en rad/s, el par generador en N·m y el ángulo de pitch en rad se almacenan en los vectores columna NomSpeedArray, NomTorqueArray y PitchAngles. La velocidad del rotor en rad/s se almacena en el vector fila RotorSpeeds.

Las matrices del espacio de estados A, B, C y D se almacenan dentro de la estructura SYSTURB (SYSTURB.A, SYSTURB.B SYSTURB.C y SYSTURB.D respectivamente) (Figura 39). En dicha estructura se incluye también el nombre de las variables de entrada (inputname), las variables de salida (outputname) y los estados del espacio de estados (statename).

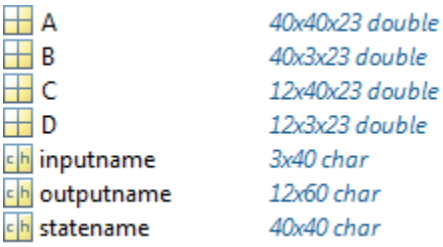

Figura 39. Ejemplo estructura SYSTURB de BLADED

Para poder trabajar con post-procesadores inicialmente diseñados para ser utilizados con BLADED y poder hacerlo con modelos lineales obtenidos con FAST, no solo es necesario emplear la misma estructura, sino también las mismas unidades y nombres de variables, tanto de entrada (Tabla 8) como de salida (Tabla 9).

	inputname	unidades
1	Collective wind speed	m/s
2	Collective pitch angle demand	rad
3	Generator torque demand	N·m

Tabla 8. Variables de entrada estandarizadas

	outputname	unidades
1	Rotor speed	rad/s
2	Generator speed	rad/s
3	Generator torque	N·m
4	Blade 1 pitch angle	rad
5	Nacelle fore-aft displacement	m
6	Nacelle fore-aft velocity	m/s
7	Nacelle x-acceleration	m/s^2
8	Blade 1 My (Principal axes), Distance along blade= 0m	N·m
9	Blade 1 Mx (Principal axes), Distance along blade= 0m	N·m
10	LSS torque	N·m
11	Tower My, Tower station height= 0m	N·m
12	Measured power	W

Tabla 9. Variables de salida estandarizadas

A lo largo del presente proyecto se van a emplear linealizaciones obtenidas con FAST y modificadas para trabajar con la estructura estandarizada de BLADED, tal y como se presentan en el apartado Modelo lineal obtenido. Sin embargo, el cambio de unidades y nombres para su total correspondencia con BLADED (ANEXO B: Estandarización LinModel) sólo se empleará en el caso de emplear las linealizaciones de FAST con otras herramientas externas.

Validación de la linealización

Uno de los principales objetivos del trabajo es la creación de una herramienta de validación del modelo lineal de FAST. Al haber empleado un formato estandarizado como salida de la linealización de FAST, el uso de dicha herramienta también puede aplicarse a modelos lineales obtenidos con otros softwares.

La validación de un modelo lineal se realiza por medio de una comparativa con el modelo real o modelo no lineal del cual parte la linealización. En este caso, como FAST permite linealizar y simular, no es necesario el uso de un software externo de simulación. Además, permite emplear el mismo archivo principal (.fst) que define la máquina, sin perturbar ninguna de sus características.

La comparativa entre la linealización y el modelo real se agrupa en:

- Lazo abierto. Se simula temporalmente la respuesta del modelo lineal ante una entrada escalón, rampa o senoidal y se compara con lo obtenido en la simulación de FAST ante la misma entrada. Al introducir entradas sinusoidales a diferentes frecuencias se recrea el diagrama de Bode del modelo real y se compara con el obtenido en la linealización.
- Lazo cerrado. Se cierran los lazos del modelo lineal empleando la función *connect* de MATLAB y se simula temporalmente su respuesta ante una entrada escalón. El modelo no lineal de FAST se cierra empleando bloques en Simulink y se analiza la respuesta del sistema ante la misma entrada escalón.

Tanto en las comparativas en lazo abierto como en las de lazo cerrado, el modelo lineal empleado corresponde a una velocidad de viento determinada mientras que el sistema simulado con FAST es global. Esto implica que debe haber diferencias entre ambos sistemas, pero la dinámica de ambos sistemas debería ser similar en el entorno del punto de equilibrio.

Para poder realizar las simulaciones con FAST y controlar las entradas así como su cierre de lazos, se implementa el bloque FAST_SFunc a partir de la función FAST_SFunc.mexw46 en Simulink. El subsistema que contiene el interfaz con FAST tiene como entradas:

- Par generador demandado en N·m.
- Potencia en W como par demandado por velocidad de giro del generador medida.
- Posición en rad y velocidad en rad/s del yaw o sistema de orientación. Ambos se fijan al valor 0.
- Ángulo de paso de pala o pitch en rad.

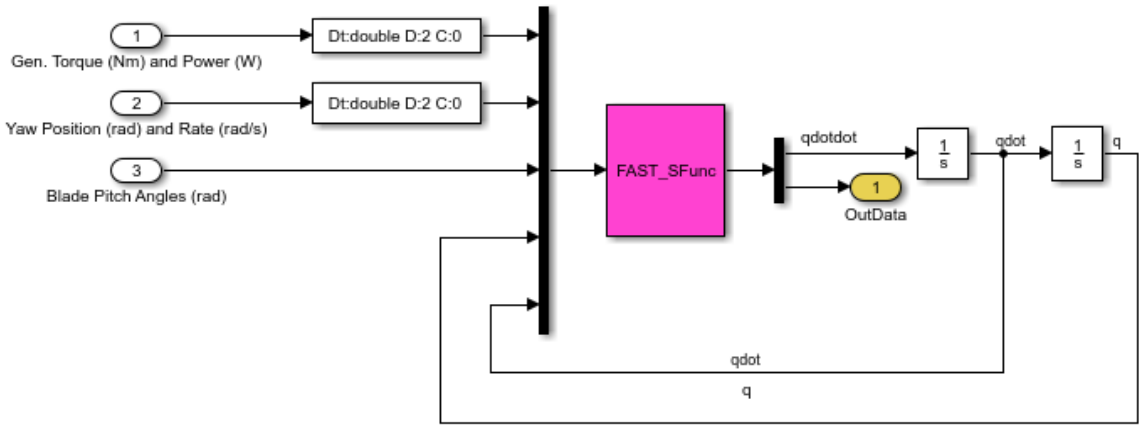

Figura 40. Implementación de FAST en Simulink

Las simulaciones del modelo lineal se realizan empleando la función de MATLAB *lsim* con la que se obtiene la respuesta temporal del sistema lineal ante una entrada arbitraria. El sistema lineal que se emplea con dicha función se corresponde con la linealización a la velocidad de viento inicial de la entrada con la que se esté validando. Por ejemplo, si se quiere validar un step de viento de 14 m/s a 16 m/s, el modelo lineal que se simulará, será el correspondiente a la linealización de FAST a 14 m/s.

Lazo abierto

Para la validación de la linealización en lazo abierto se ha empleado el script del ANEXO D: Validación Lazo Abierto y la implementación de Simulink de la Figura 41.

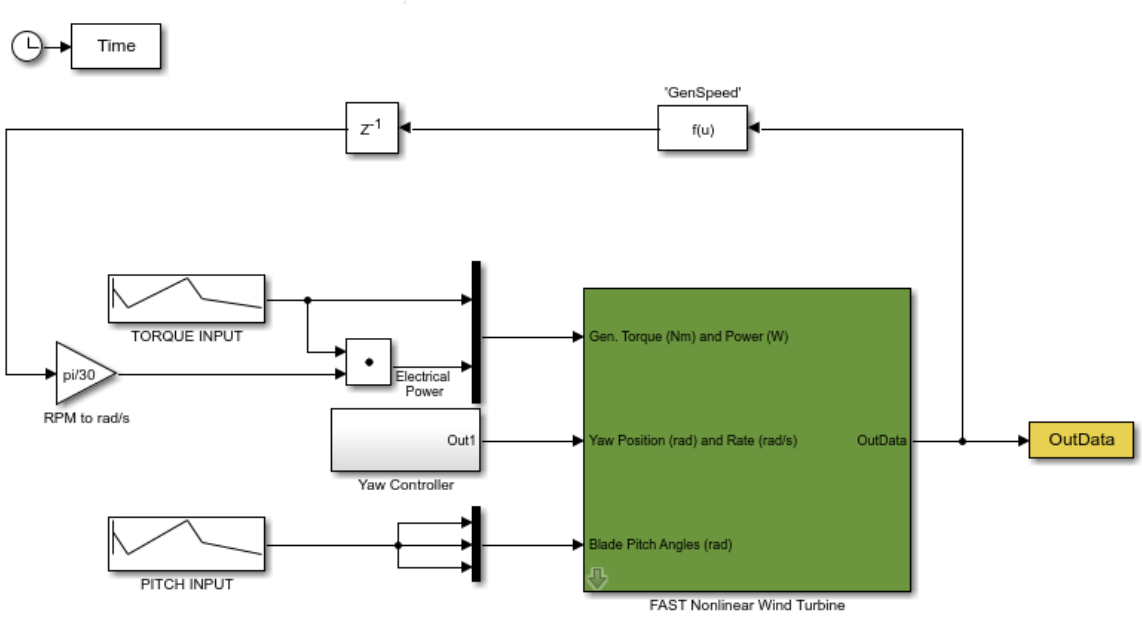

Figura 41. Esquema Simulink lazo abierto

En lazo abierto se introducen entradas del tipo escalón (step), rampa (ramp) y senoidal (sin). Debido a que el sistema tiene tres entradas (viento, pitch y par), se simula para cada entrada cada tipo de entrada. En la siguiente tabla se recogen las principales validaciones realizadas.

Tipo de entrada	Entrada	Amplitud	Velocidades de viento	Otras características
Escalón $t_{step} = 200s$ $t_{end} = 400s$	Viento	1 m/s	4 7 11 16 22	Todas las zonas
	Pitch	0.02 rad	13 16 19 22	Zona 3. Control de pitch
	Par	1000 N	4 6 8 10	Zona 1.5, 2 y 2.5. Control de par
Rampa $t_{end} = 600s$	Viento	3 m/s	4 7 11 16 22	Todas las zonas
	Pitch	0.1 rad	13 16 19 22	Zona 3. Control de pitch
	Par	2000 N	4 6 8 10	Zona 1.5, 2 y 2.5. Control de par

Senoidal $f = 0.01$ $t_{end} = 600s$	Viento	1 m/s	4 7 11 16 22	Todas las zonas
	Pitch	0.02 rad	13 16 19 22	Zona 3. Control de pitch
	Par	1000 N	4 6 8 10	Zona 1.5, 2 y 2.5. Control de par

Tabla 10. Validaciones lazo abierto

Escalón

Se introduce un **escalón de viento** de amplitud 1 m/s cuando la máquina está trabajando a 4, 7, 11, 16 y 22 m/s (Figura 42). Cuando la máquina está trabajando a 4 m/s, el modelo lineal se acelera rápidamente y no alcanza el régimen estacionario (Figura 42). Este comportamiento es normal ya que un escalón de amplitud 1 m/s es porcentualmente mayor cuando se hace a partir de 4 m/s (25%), que cuando se hace a partir de 22 m/s (4.5%).

El modelo lineal que se emplea en cada escalón corresponde con el modelo de la turbina de la velocidad de viento inicial, por ello el modelo lineal difiere del no lineal una vez aplicado el escalón. Tanto a 7 m/s como a 11 m/s hay diferencias notables en el punto de equilibrio tanto antes del escalón como después (Figura 43Figura 42 y Figura 44). Aun así, la dinámica de ambos modelos es similar.

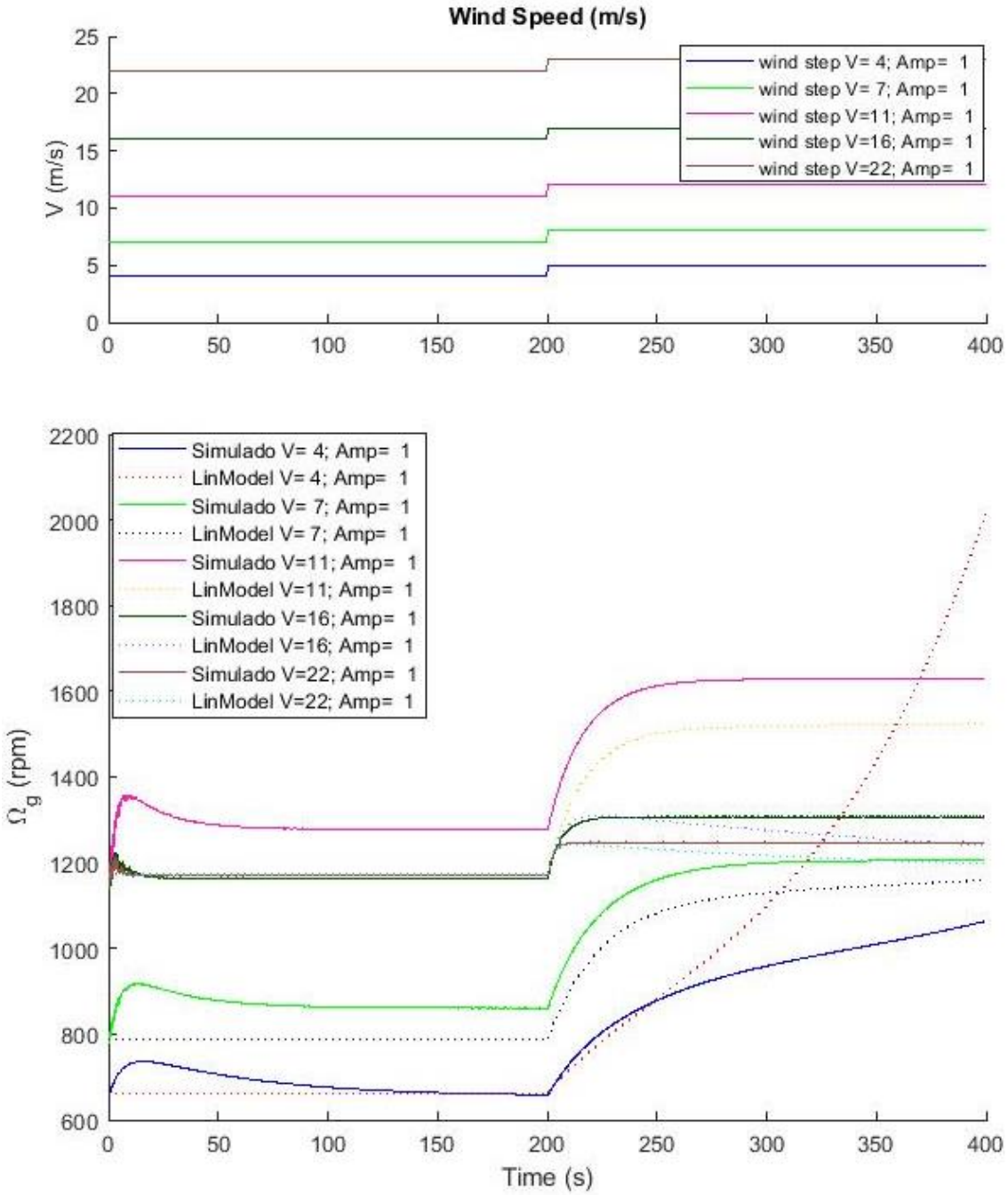

Figura 42. Escalón de viento. Ω_g

Según la velocidad de viento con la que se simule, se impone un par y un pitch de entrada, tanto en el modelo lineal como en el modelo no lineal, correspondientes al valor encontrado como punto de equilibrio en el modelo lineal (Figura 43).

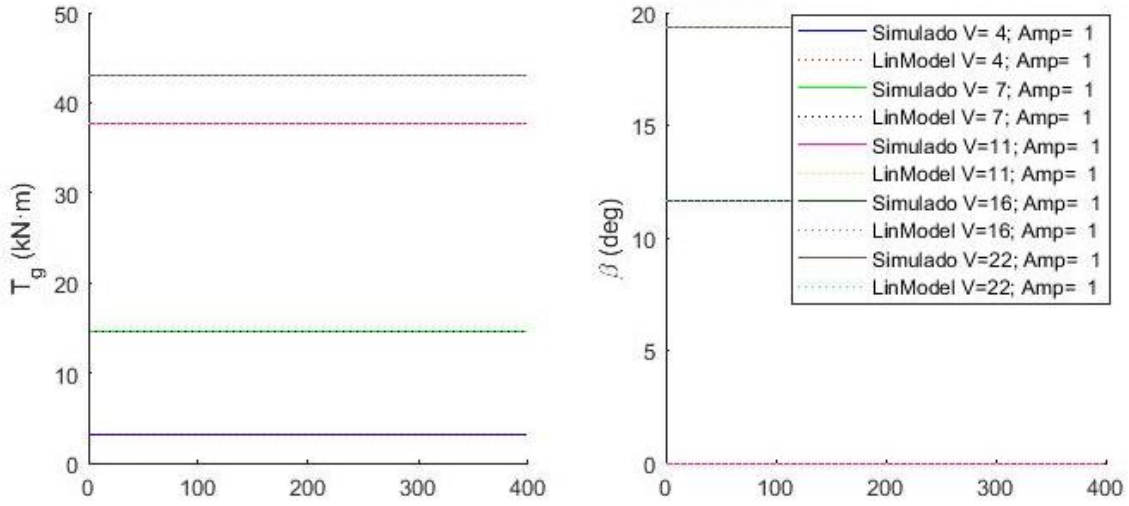

Figura 43. Escalón de viento. Tg y β

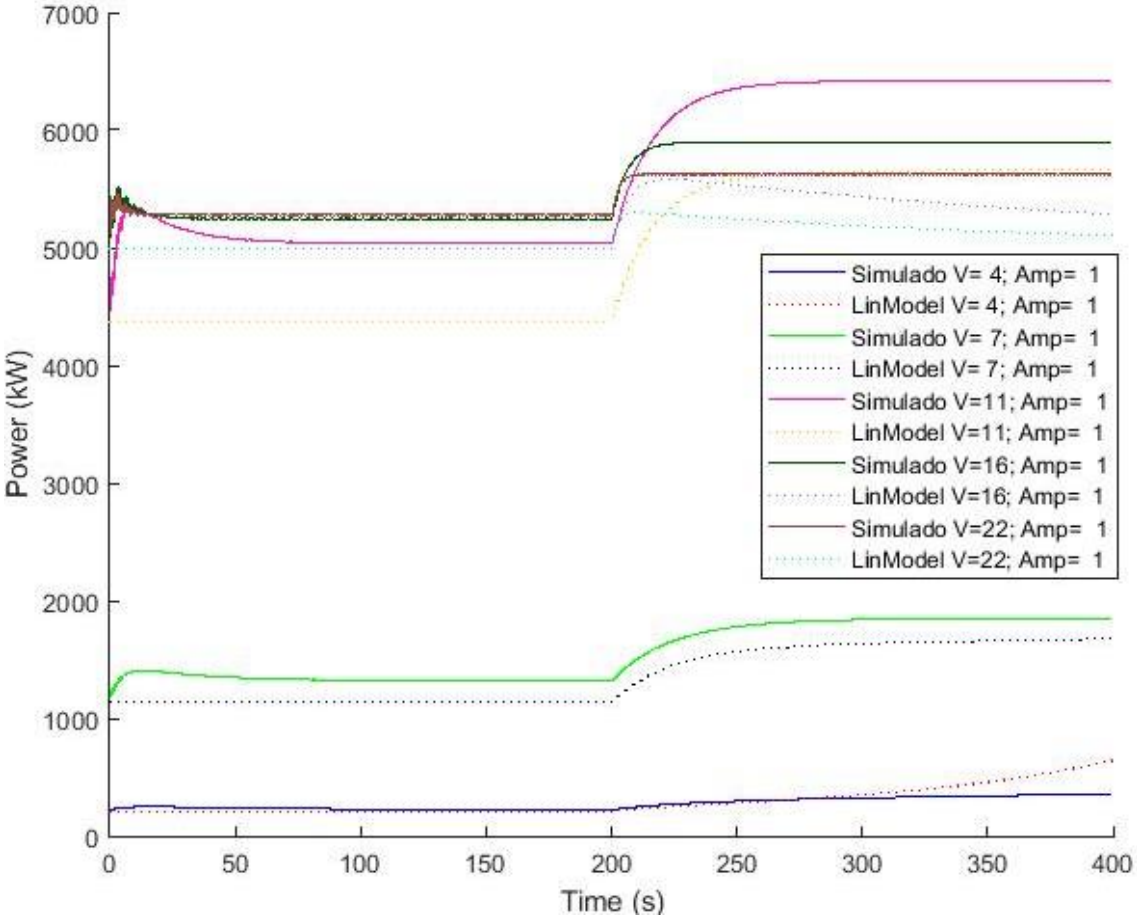

Figura 44. Escalón de viento. Potencia

Se introduce un **escalón de pitch** de amplitud 0.02 radianes cuando la máquina está trabajando a 13, 16, 19 y 22 m/s (Figura 45). Al analizarse únicamente en zona nominal al ser la zona en la que se trabaja con control de pitch, el par se establece su valor nominal (Figura 46).

La dinámica de ambos modelos es similar y la variación del punto de equilibrio entre ambos no es tan acusada como en el anterior caso. Tras 25 segundos del escalón de pitch, el modelo lineal

se desestabiliza en vez de mantenerse en un nuevo punto de equilibrio (Figura 45 y Figura 47). Sin embargo, en condiciones normales el pitch no es un escalón, sino una señal proporcionada por un controlador, por lo que con que la dinámica justo tras el escalón sea similar, es suficiente para el correcto diseño del controlador.

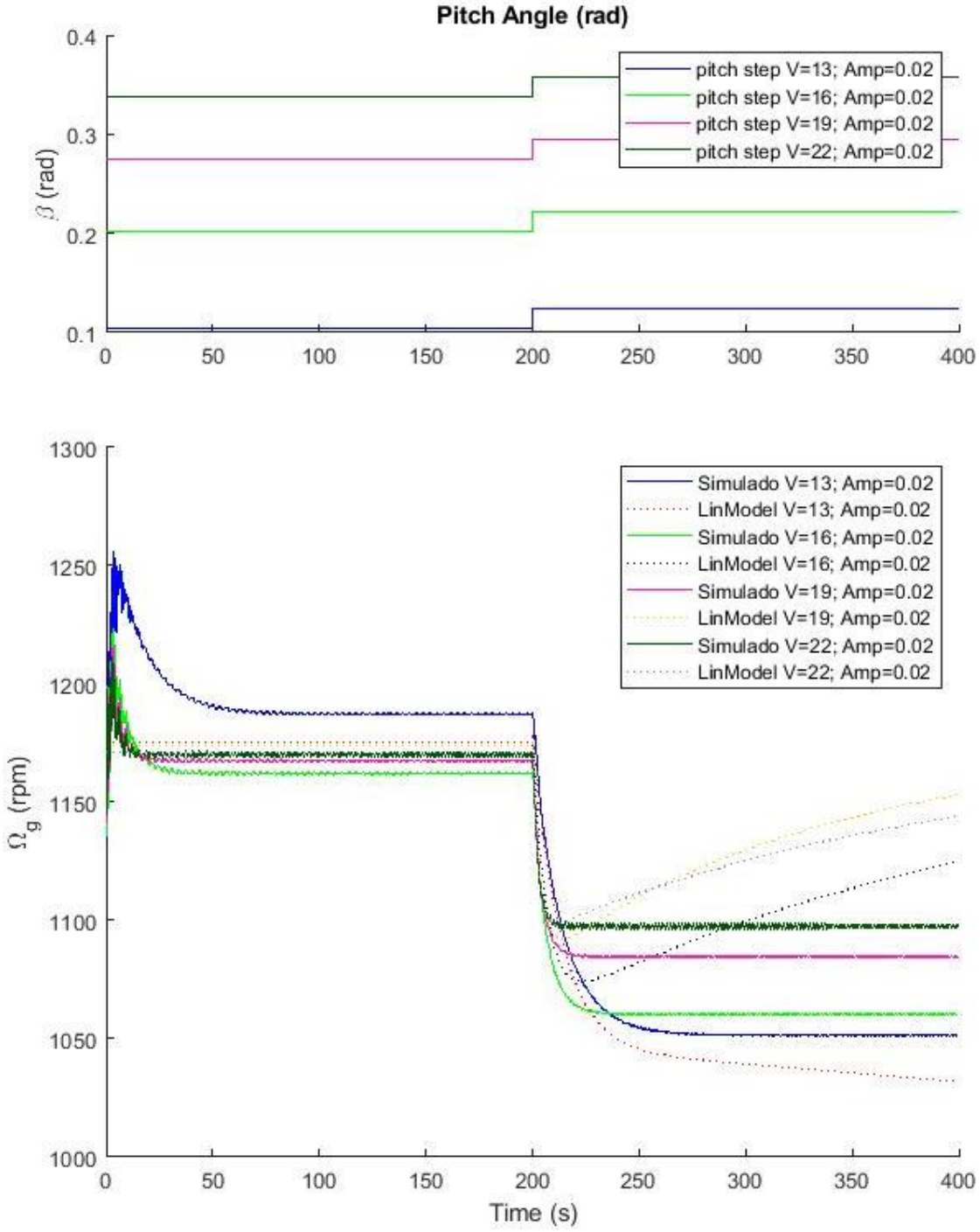

Figura 45. Escalón de pitch. Ω_g

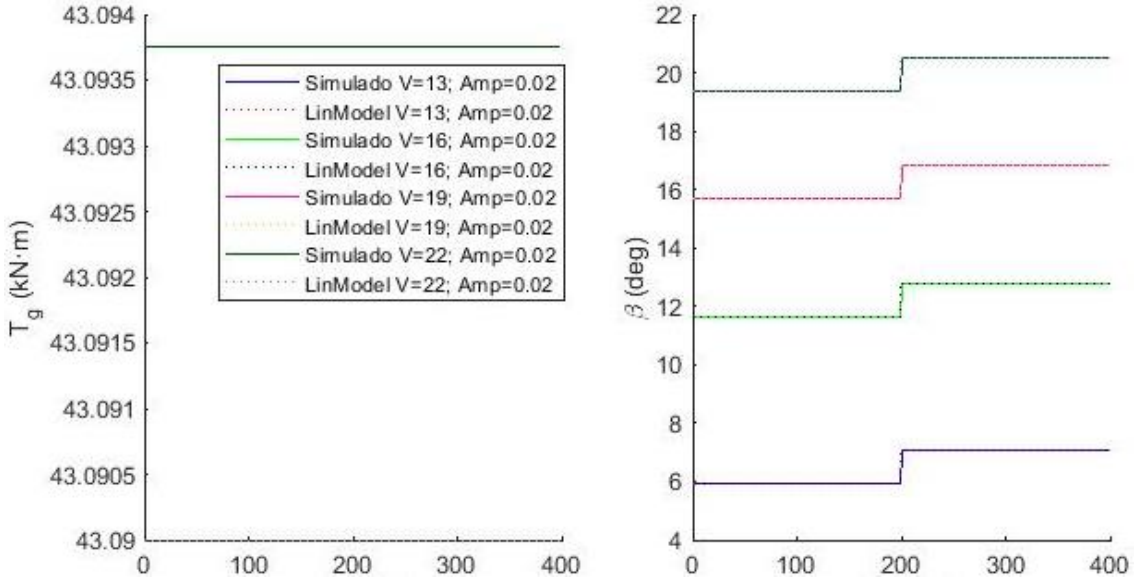

Figura 46. Escalón de pitch. Tg y β

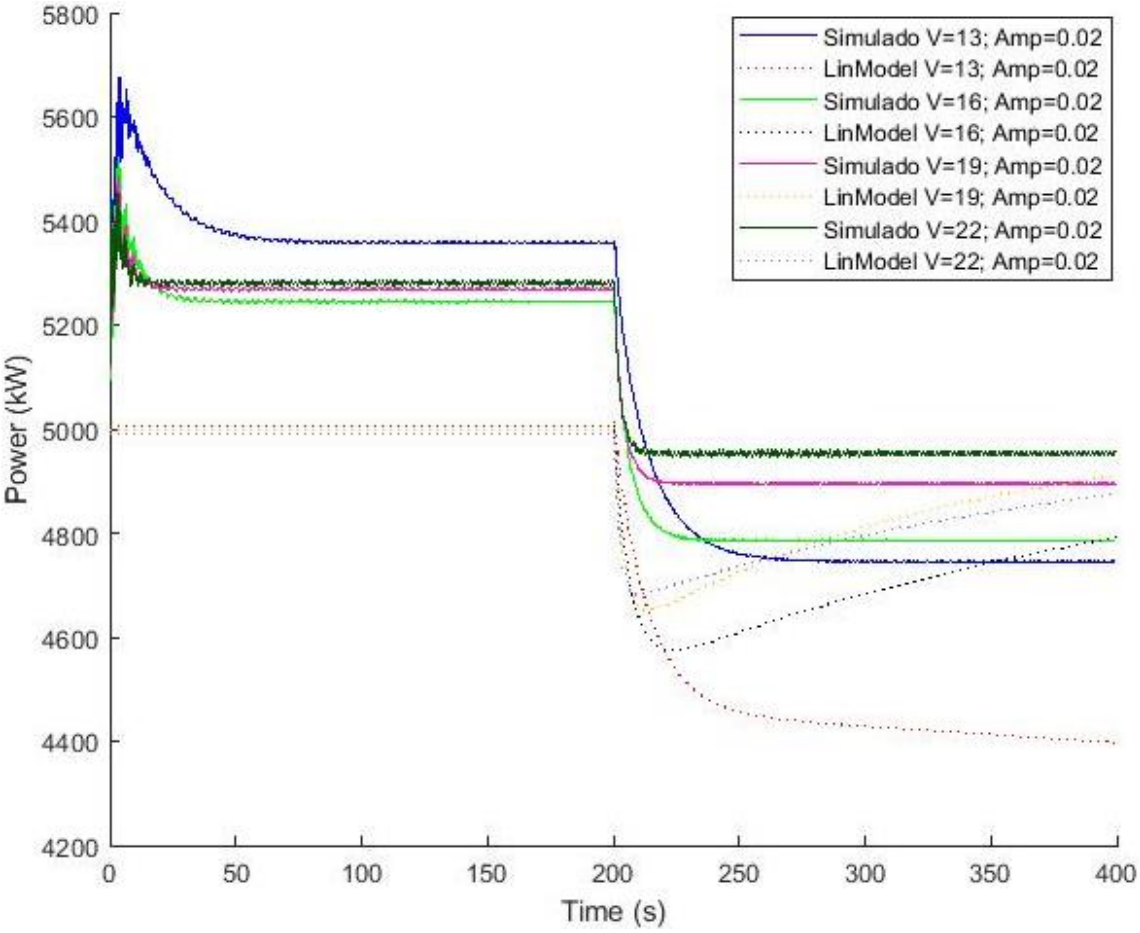

Figura 47. Escalón de pitch. Potencia

Se introduce un **escalón de par** de amplitud 1000 N·m cuando la máquina está trabajando a 4, 6, 8 y 10 m/s (Figura 48Figura 42). Cuando la máquina está trabajando a 4 m/s, el modelo lineal se frena rápidamente y no alcanza el régimen estacionario. Este comportamiento es normal ya que un escalón de amplitud 1000 N·m ante un par de equilibrio de 3.5 kN·m correspondiente a

un viento de 4 m/s es porcentualmente mucho mayor que un par de equilibrio de 30 kN·m correspondiente a un viento de 10 m/s (Figura 49).

Al igual que en los anteriores casos, el punto de equilibrio entre el modelo lineal y el no lineal difieren, especialmente a 6, 8 y 10 m/s. Sin embargo la dinámica inicial tras el escalón es similar en ambos modelos (Figura 48 y Figura 50).

El pitch demandado se establece con valor 0 (Figura 49) debido a que los escalones de par han sido introducidos en las zonas 1.5, 2 y 2.5 en las que hay control de par y no de pitch.

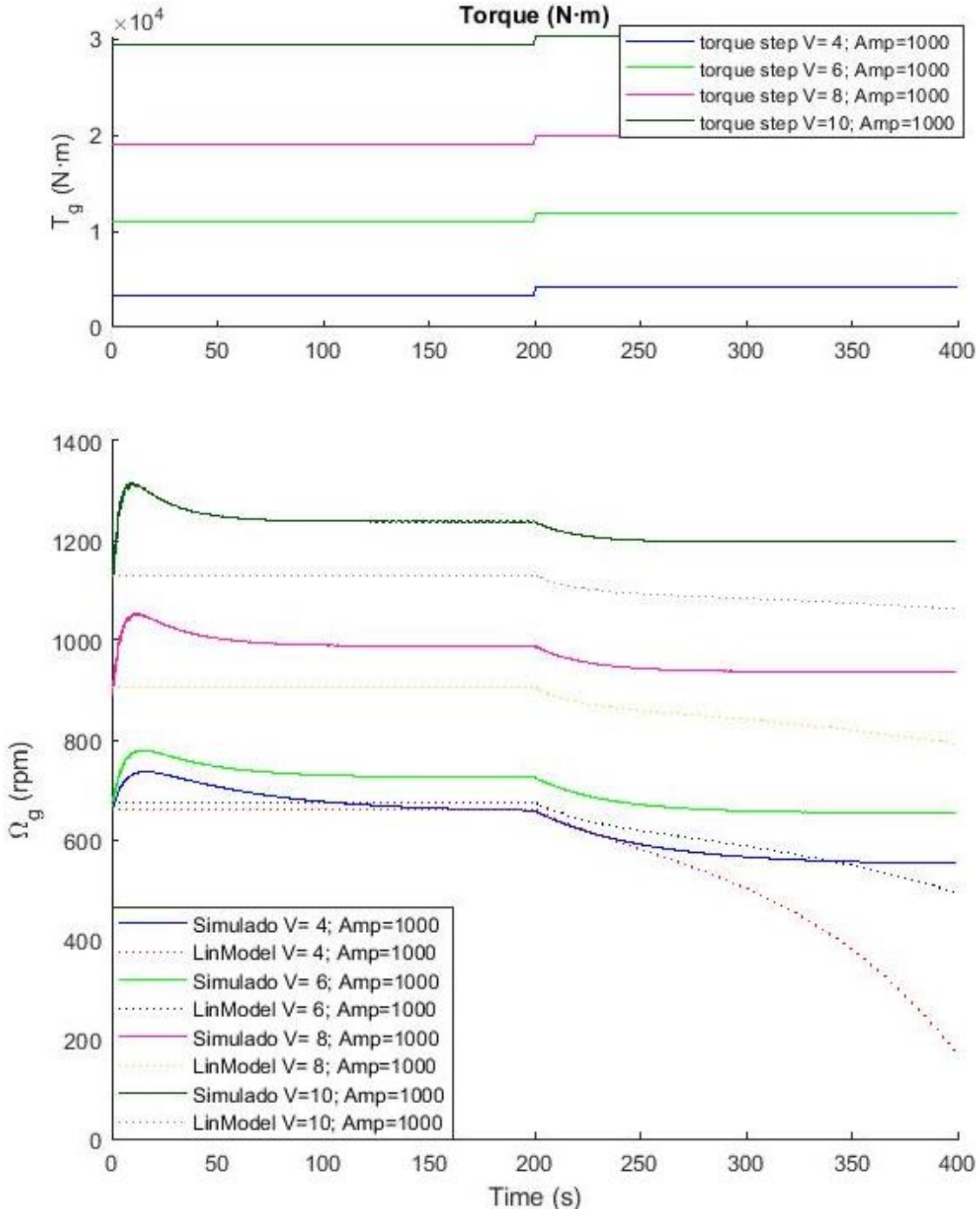

Figura 48. Escalón de par. Ω_g

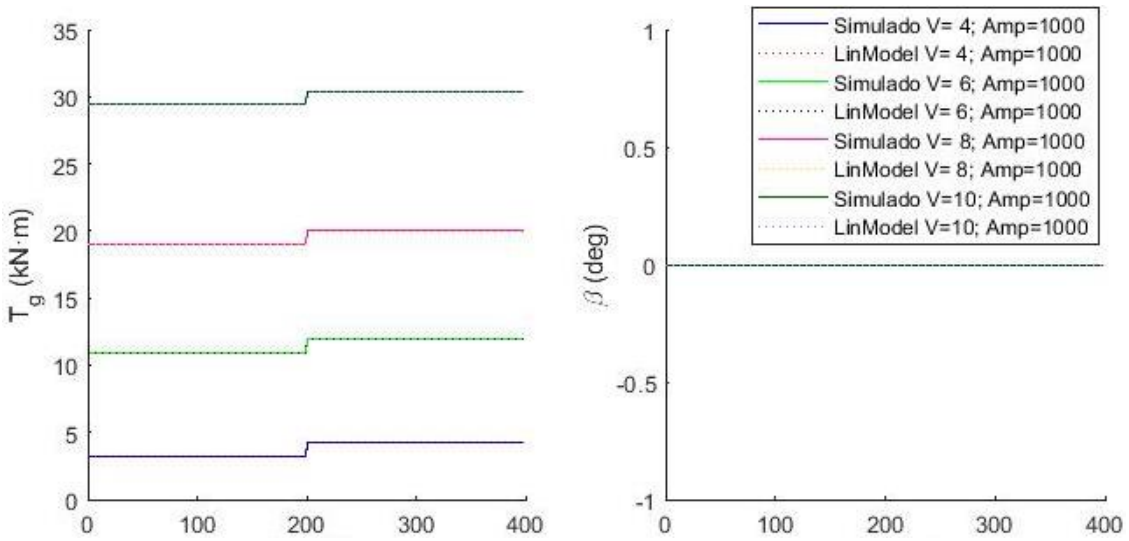

Figura 49. Escalón de par. Tg y β

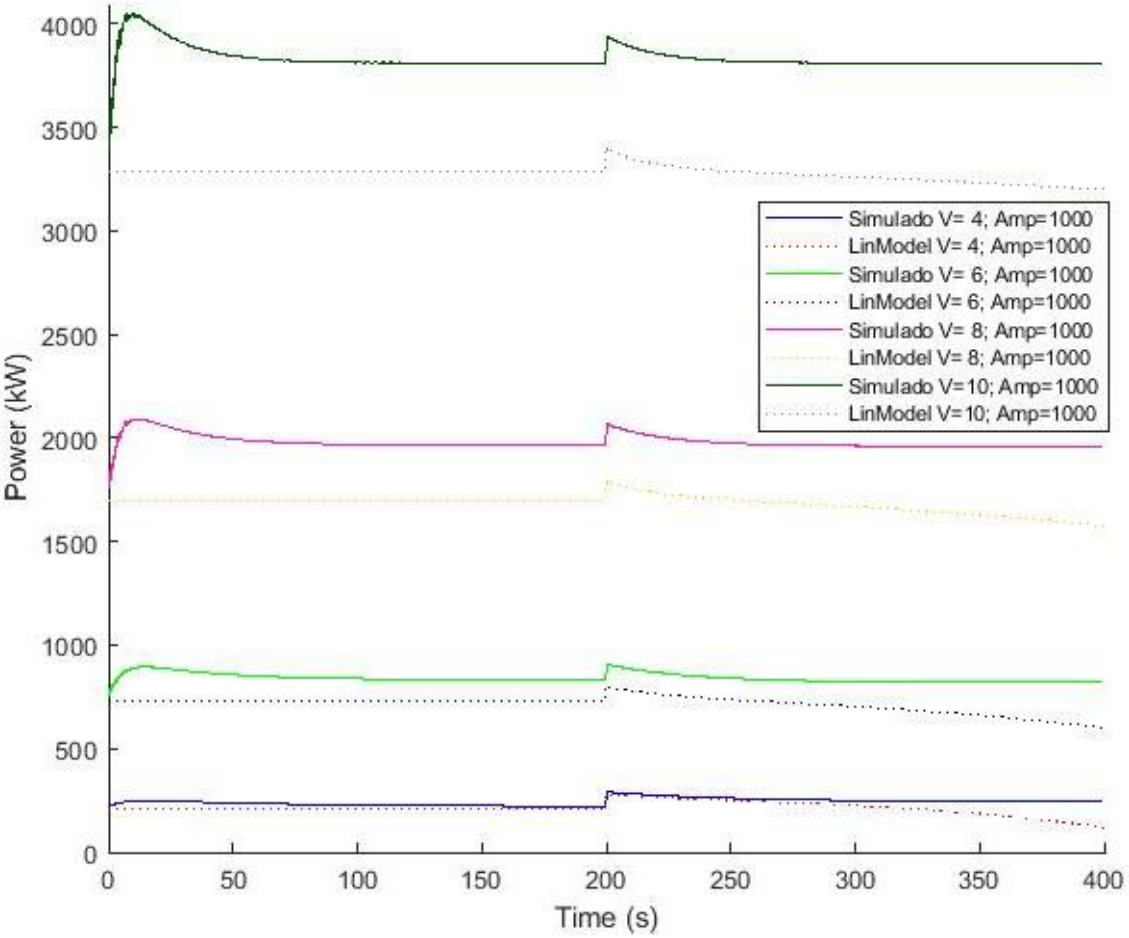

Figura 50. Escalón de par. Potencia

Rampa

Se introduce una **rampa de viento** de amplitud 3 m/s en 600 s (10 min) cuando la máquina está trabajando a 4, 7, 11, 16 y 22 m/s (Figura 51Figura 42). Cuando la máquina está trabajando a 4 m/s, tanto el modelo lineal como el no lineal se aceleran, aunque especialmente el modelo lineal (Figura 53). Este comportamiento es normal ya que una rampa de amplitud 3 m/s es

porcentualmente mayor cuando se hace a partir de 4 m/s (75%), que cuando se hace a partir de 22 m/s (13.6%).

El modelo lineal que se emplea en cada rampa corresponde con el modelo de la turbina de la velocidad de viento inicial, por ello el modelo lineal difiere del no lineal aunque la dinámica es similar. Tanto a 7 m/s como a 11 m/s hay diferencias en el punto de equilibrio inicial, al igual que lo había en el escalón de viento.

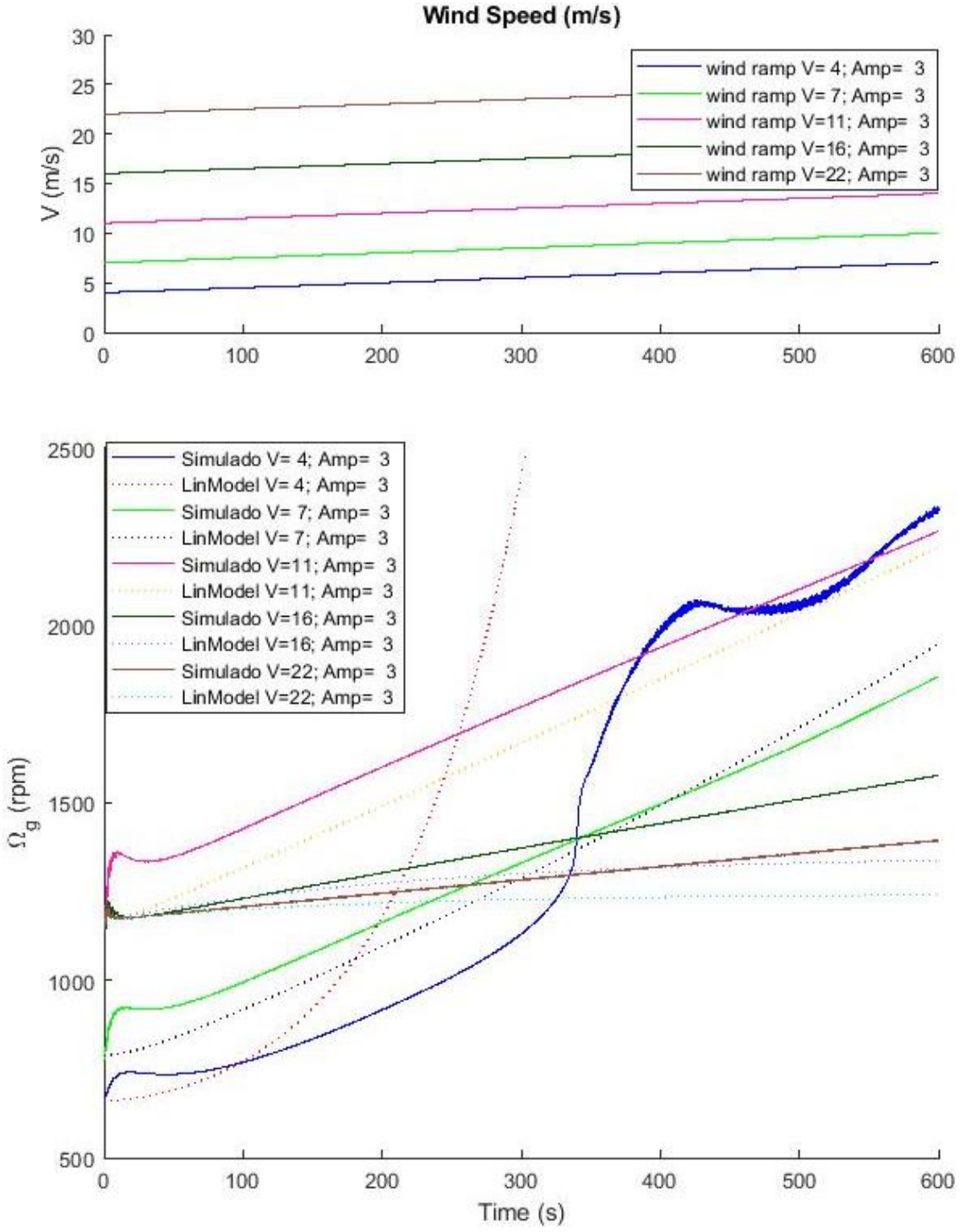

Figura 51. Rampa de viento. Ω_g

Según la velocidad de viento con la que inicie la simulación, se impone un par y un pitch de entrada, tanto en el modelo lineal como en el modelo no lineal, correspondientes al valor encontrado como punto de equilibrio en el modelo lineal (Figura 52).

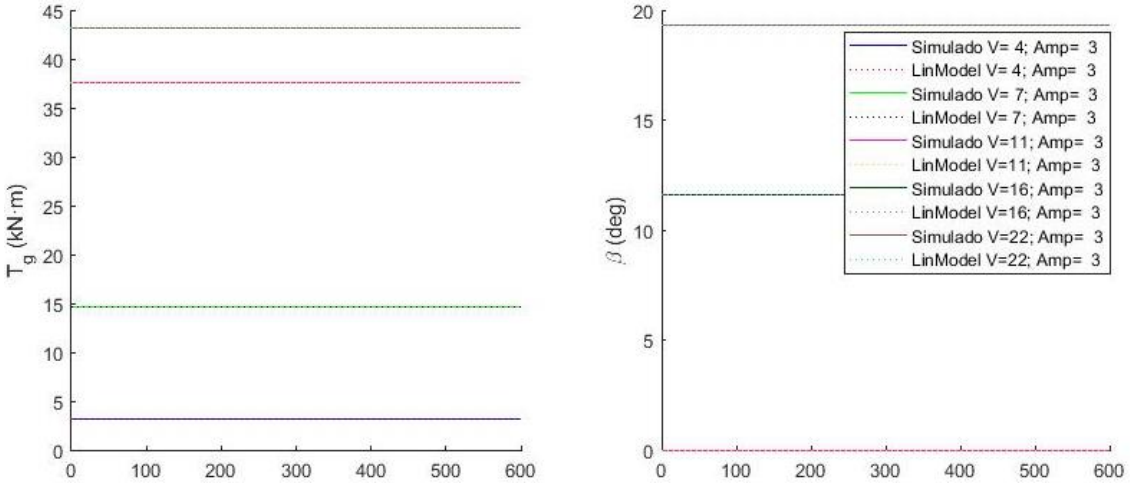

Figura 52. Rampa de viento. Tg y β

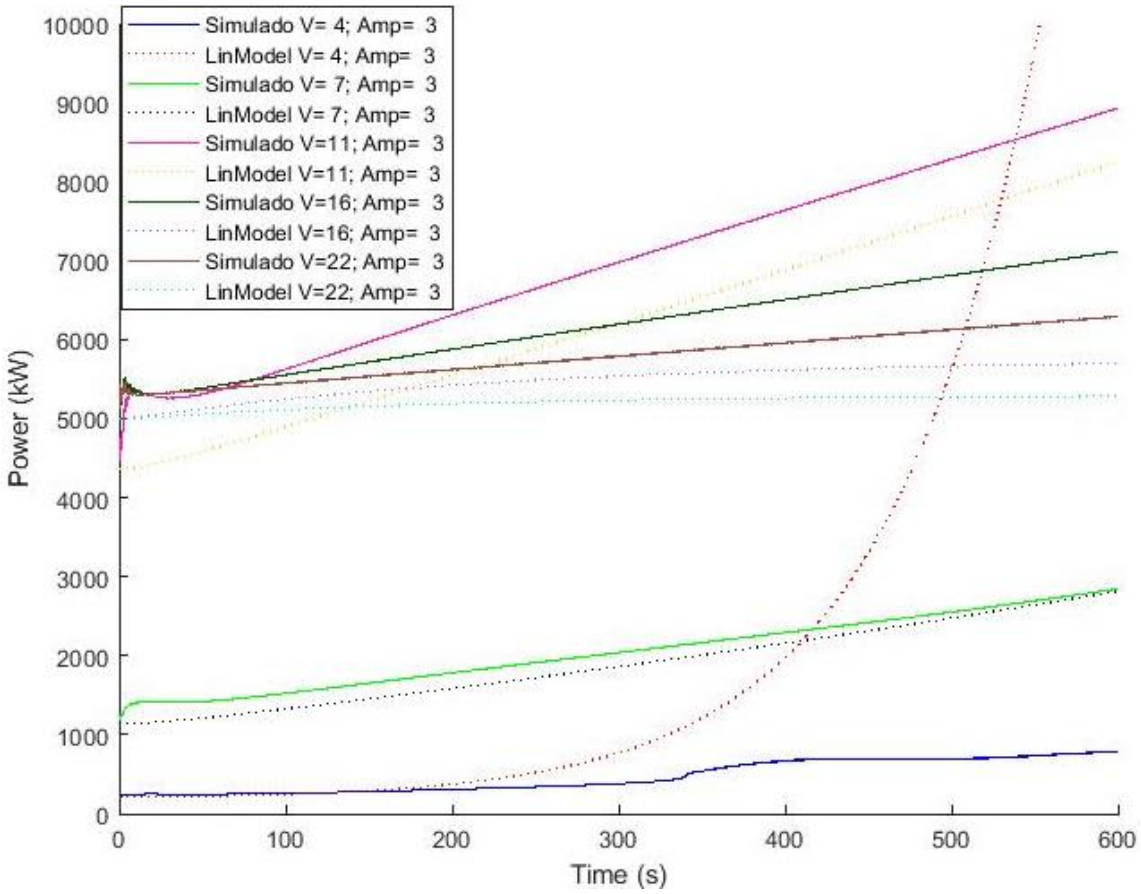

Figura 53. Rampa de viento. Potencia

Se introduce una **rampa de pitch** de amplitud 0.1 radianes cuando la máquina está trabajando a 13, 16, 19 y 22 m/s (Figura 54). Al analizarse únicamente en zona de operación nominal al ser la zona en la que se trabaja con control de pitch, el par se establece su valor nominal (Figura 55).

La dinámica de ambos modelos es similar y aunque halla pequeñas diferencias en el punto de equilibrio, el comportamiento es el esperado ya que al aumentar el pitch y mantener el viento y el par, la máquina se frena. Tanto en el modelo lineal como en el no lineal, el sistema reduce su velocidad de giro (Figura 54) y su potencia (Figura 56) a 13 m/s más rápido que a otras velocidades de viento al ser proporcionalmente una rampa de pitch mayor.

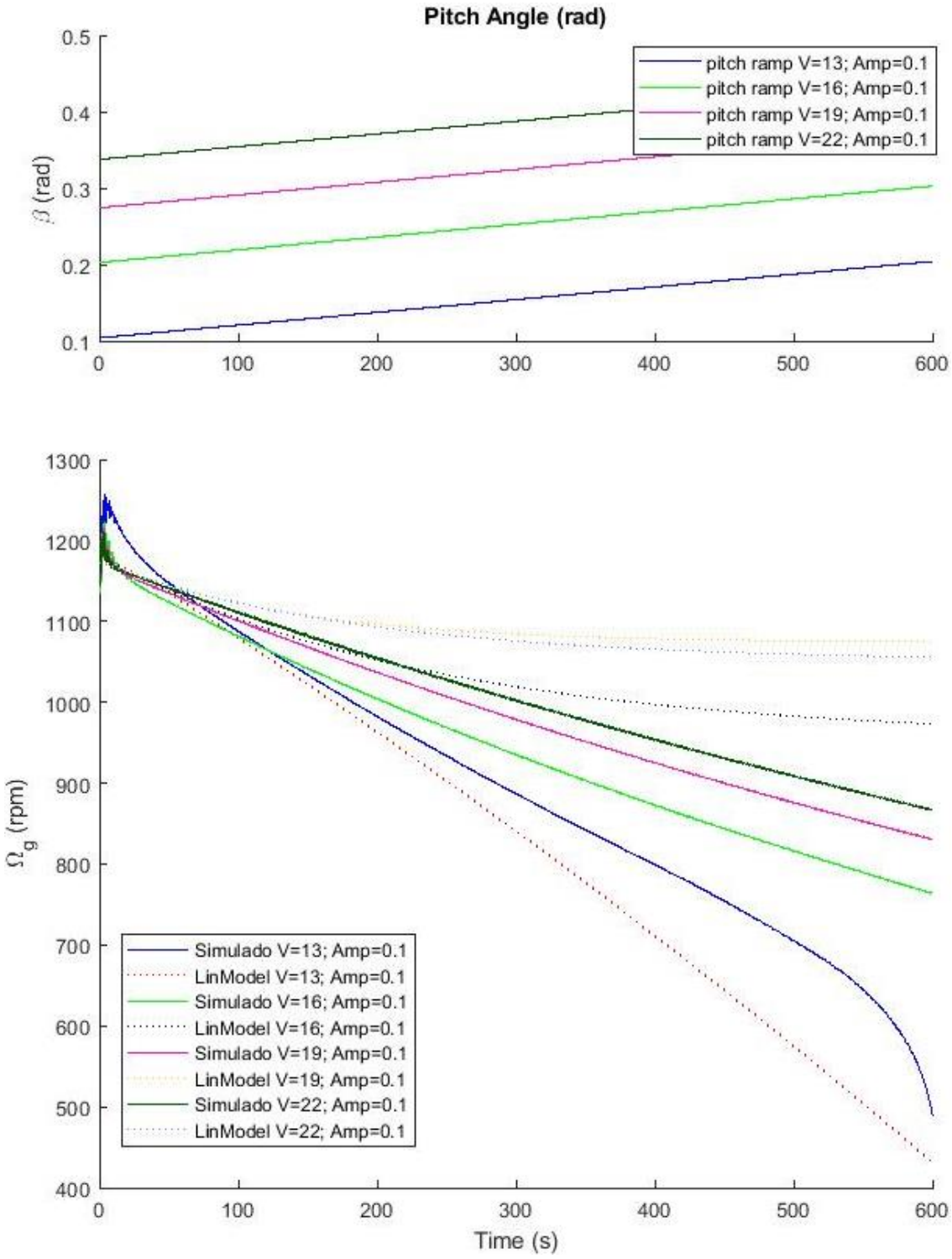

Figura 54. Rampa de pitch. Ω_g

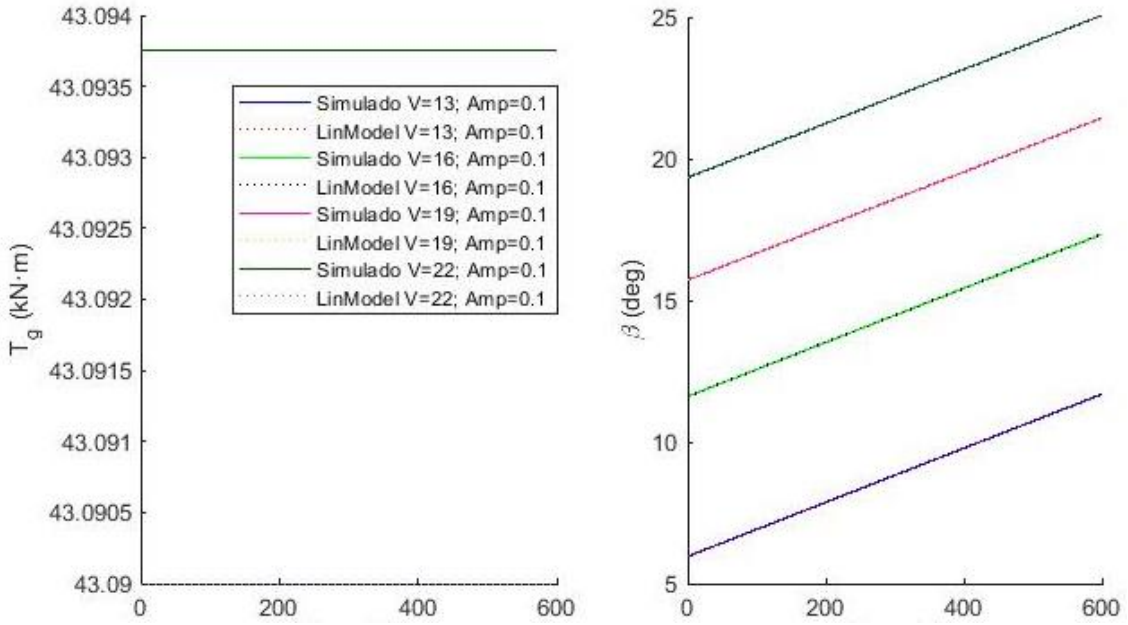

Figura 55. Rampa de pitch. Tg y β

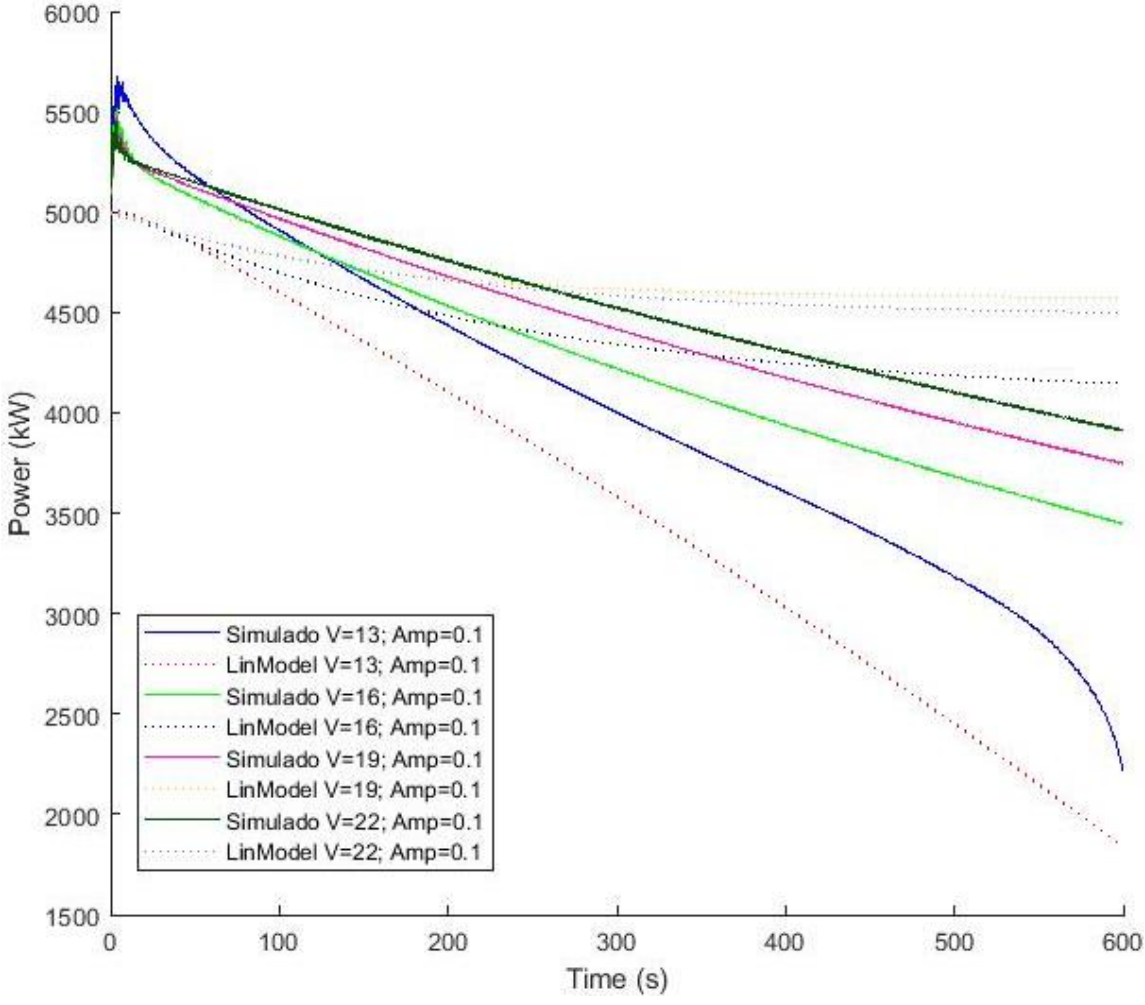

Figura 56. Rampa de pitch. Potencia

Se introduce una **rampa de par** de amplitud 2000 N·m cuando la máquina está trabajando a 4, 6, 8 y 10 m/s (Figura 57Figura 42). Cuando la máquina está trabajando a 4 m/s y 6 m/s, el modelo lineal se frena rápidamente y no alcanza el régimen estacionario. Este comportamiento es normal ya que un escalón de amplitud 2000 N·m ante un par de equilibrio de 3.5 kN·m inicial correspondiente a un viento de 4 m/s y 11 kN·m a 6 m/s es porcentualmente mucho mayor que un par de equilibrio de 30 kN·m correspondiente a un viento de 10 m/s (Figura 58).

Al igual que en los anteriores casos, el punto de equilibrio entre el modelo lineal y el no lineal difieren, especialmente a 6, 8 y 10 m/s. Sin embargo la dinámica inicial de la rampa es similar en ambos modelos (Figura 57 y Figura 59).

El pitch demandado se establece con valor 0 (Figura 58) debido a que las rampas de par han sido introducidas en las zonas 1.5, 2 y 2.5 en las que hay control de par y no de pitch.

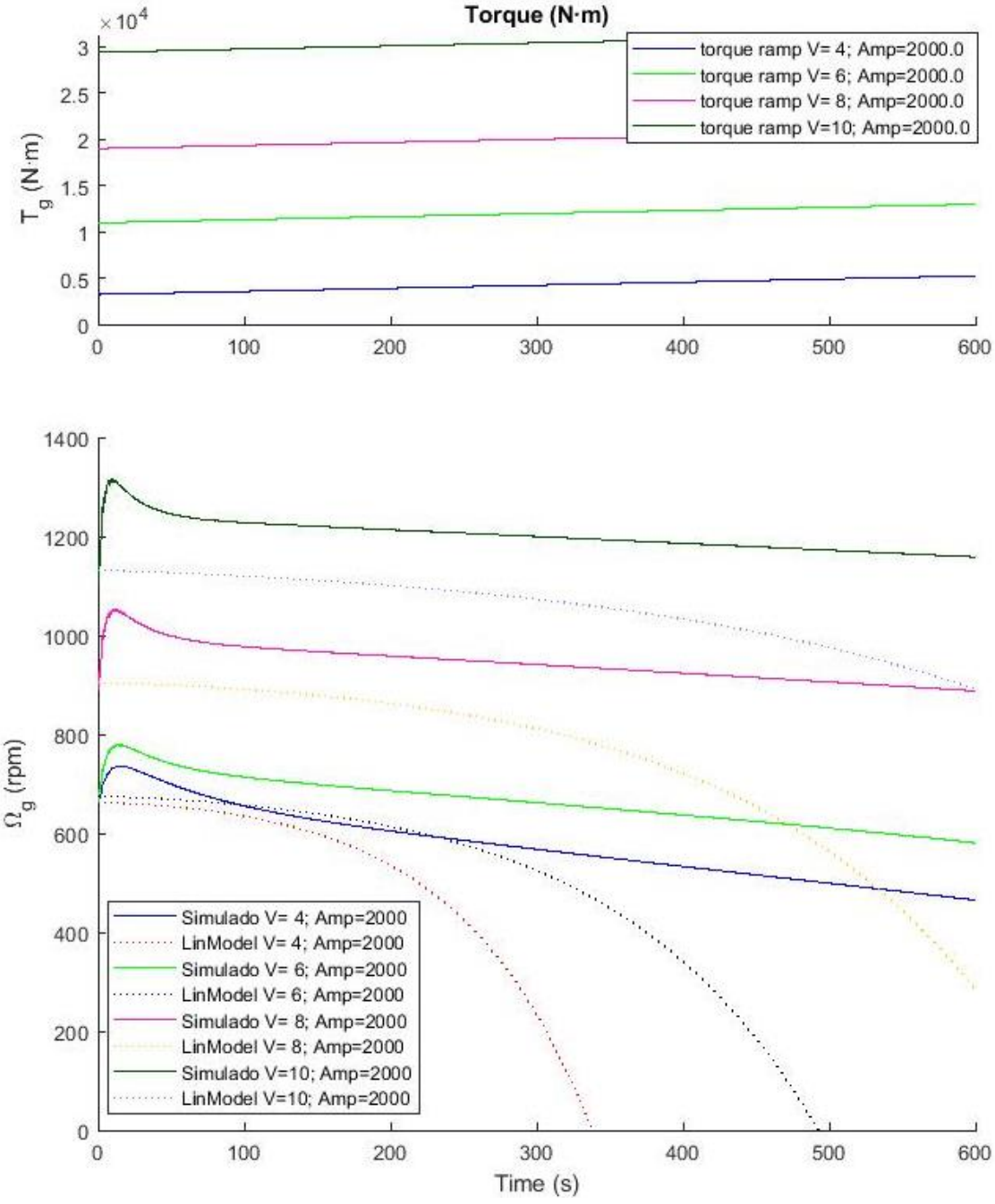

Figura 57. Rampa de par. Ω_g

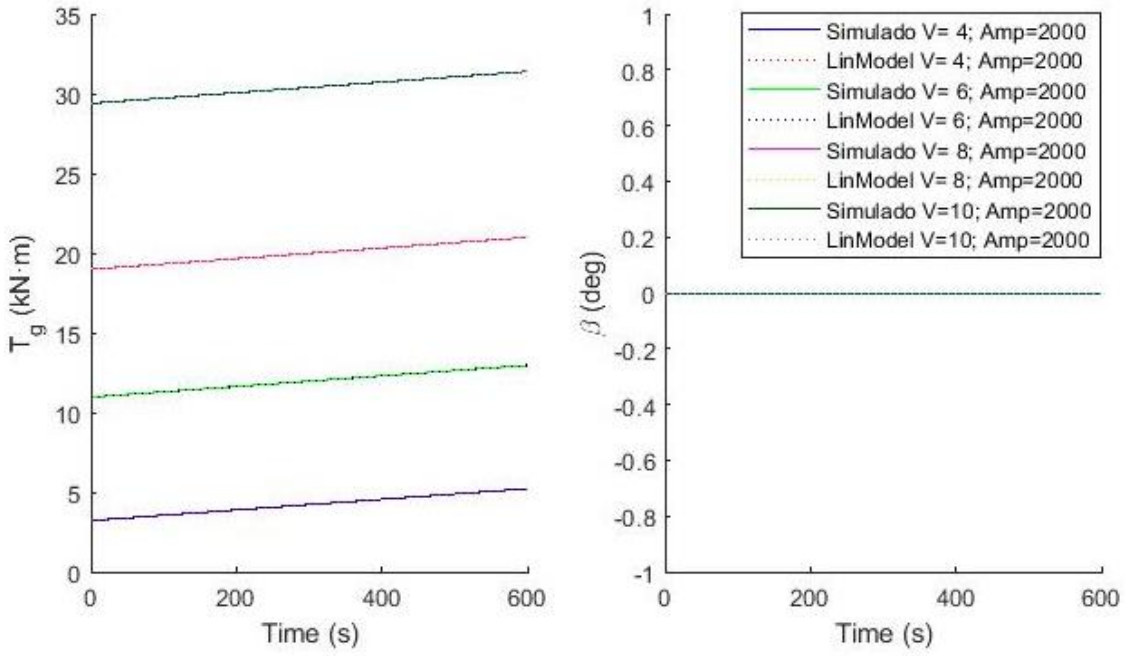

Figura 58. Rampa de par. Tg y β

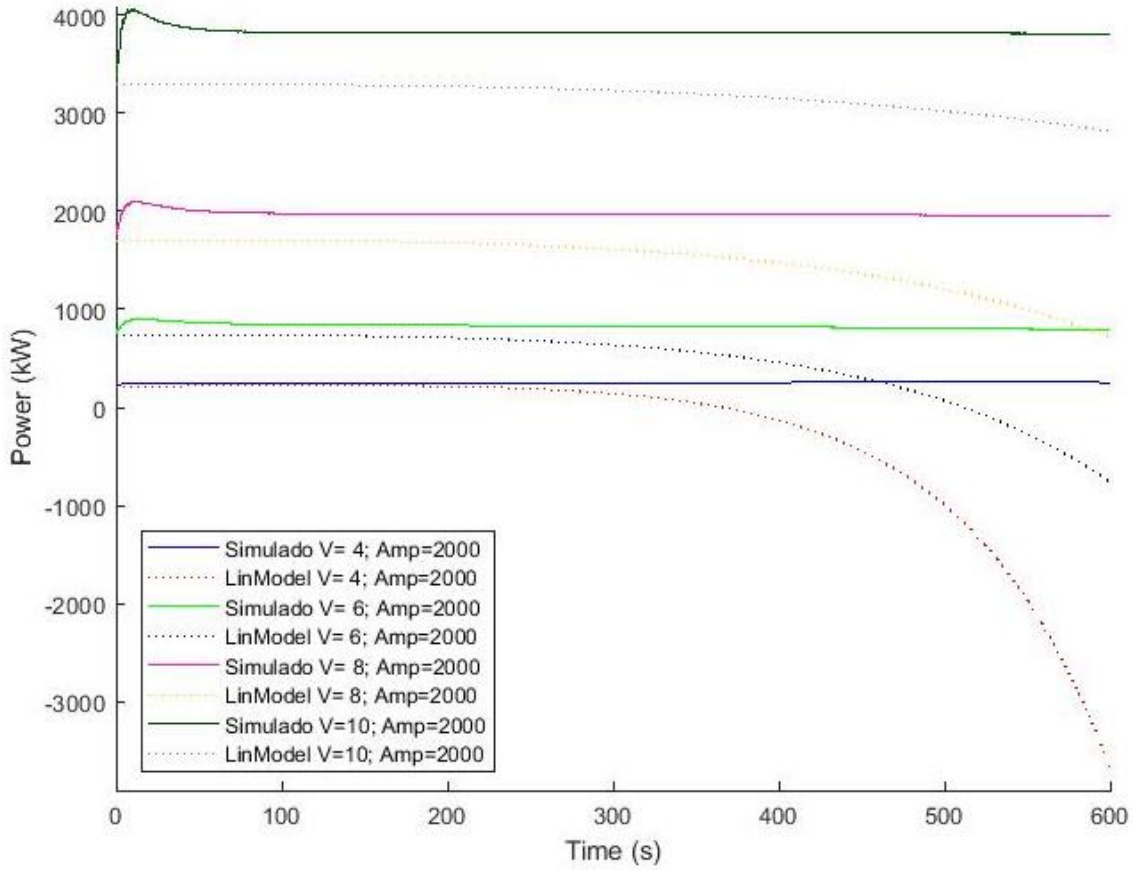

Figura 59. Rampa de par. Potencia

Sinusoide

Se introduce un **seno de viento** de amplitud 1 m/s cuando la máquina está trabajando a 4, 7, 11, 16 y 22 m/s (Figura 60Figura 42). Cuando la máquina está trabajando a 4 m/s, el modelo lineal

se inestabiliza y la máquina se acelera. Este comportamiento es normal ya que un seno de amplitud 1 m/s con una velocidad media de 4 m/s es porcentualmente mucho mayor que la misma amplitud con una velocidad media de 22 m/s.

Al igual que en los anteriores casos, el punto de equilibrio entre el modelo lineal y el no lineal difieren, aunque la dinámica de ambos modelos es similar tanto en fase como en magnitud (Figura 60 y Figura 62).

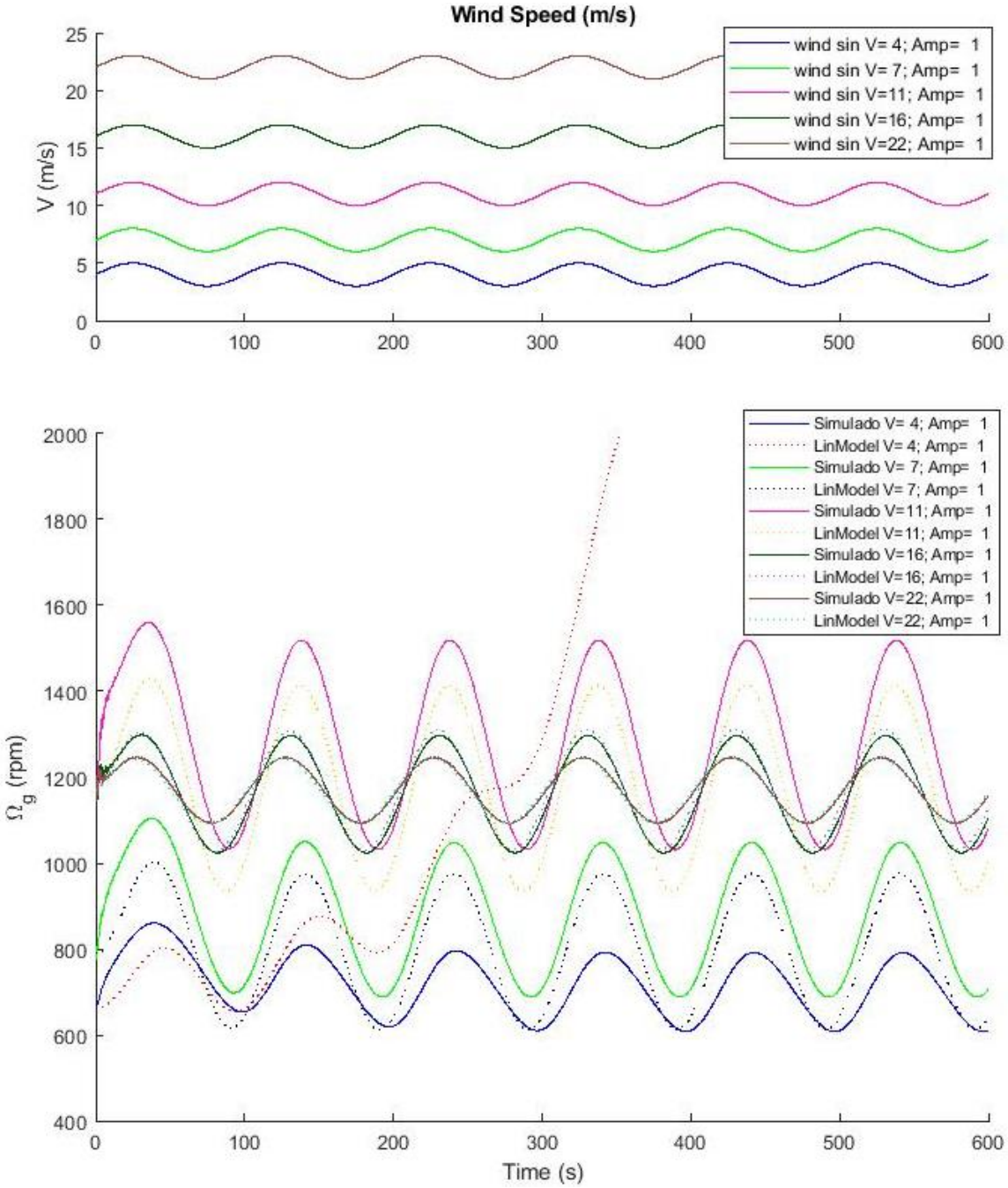

Figura 60. Seno de viento. Ω_g

El pitch y el par demandados se establecen a un valor constante correspondiente al valor del modelo lineal para la velocidad de viento media de cada uno de los senos (Figura 61).

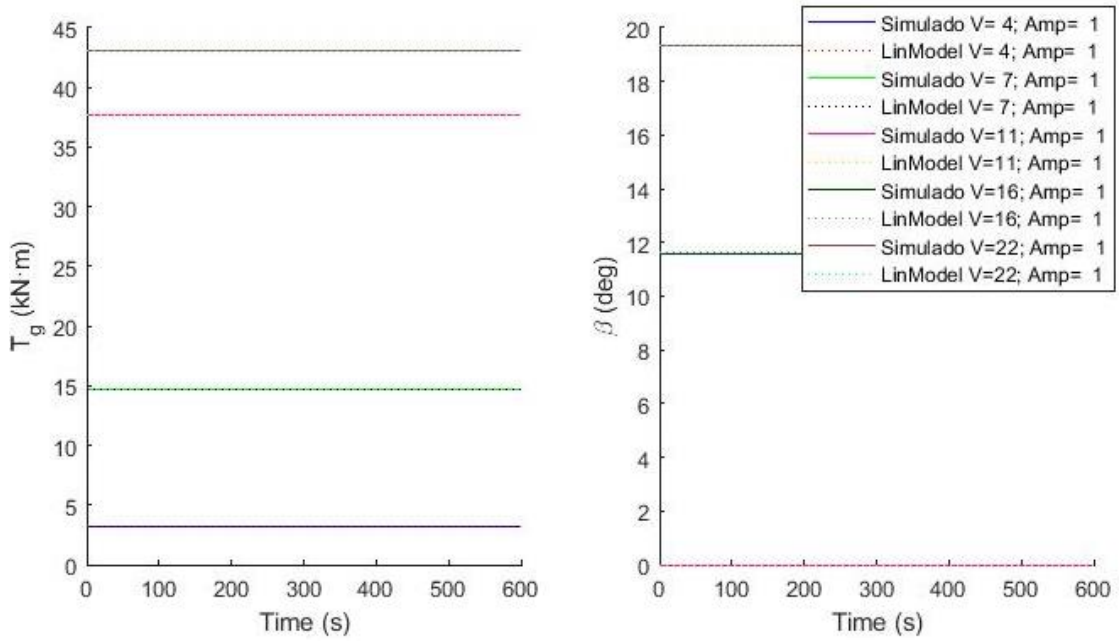

Figura 61. Seno de viento. Tg y β

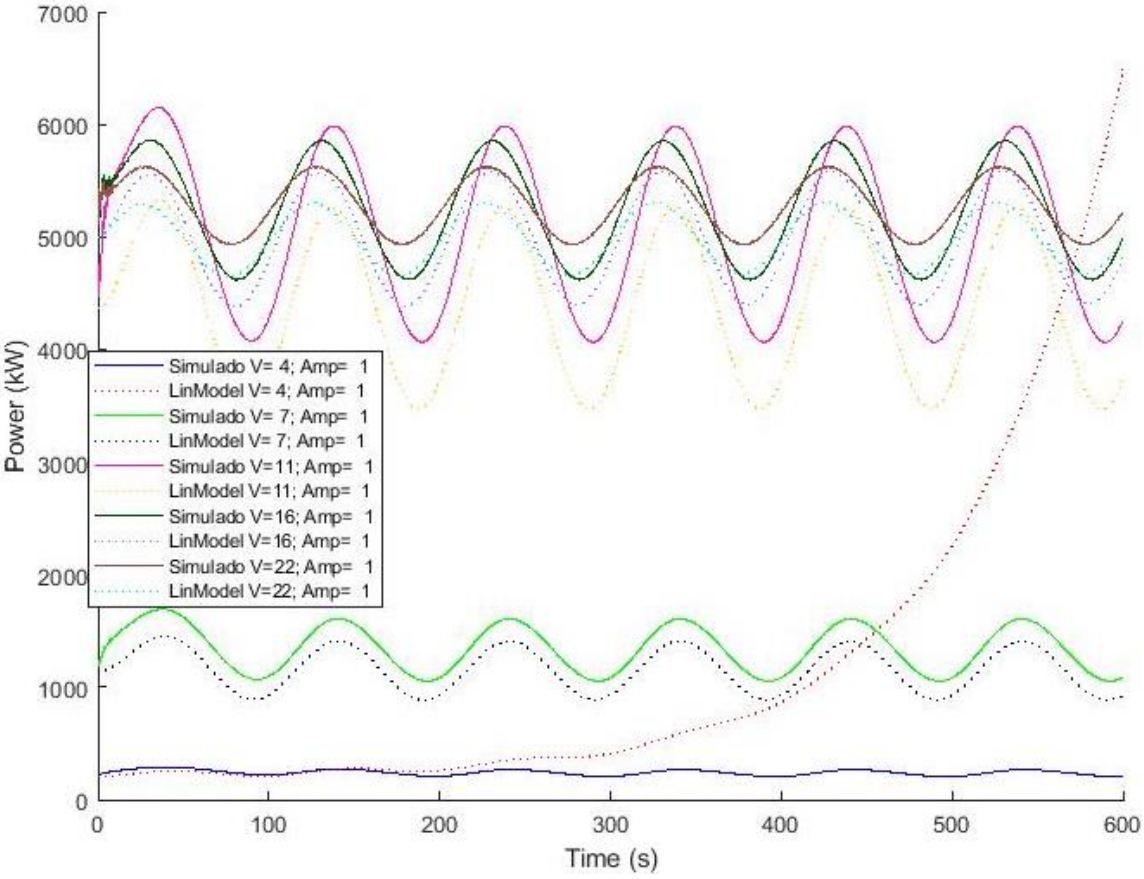

Figura 62. Seno de viento. Potencia

Se introduce un **seno de pitch** de amplitud 0.02 radianes cuando la máquina está trabajando a 13, 16, 19 y 22 m/s (Figura 63). Solo se analiza la zona nominal ya que es la zona de operación en la que se emplea el control de pitch. Por ello, el par se establece su valor nominal mientras que el valor del ángulo de pitch tiene forma senoidal (Figura 64).

La dinámica de ambos modelos es similar y aunque halla pequeñas diferencias en el punto de equilibrio (Figura 65), tanto la fase como la ganancia tienen valores cercanos para ambos modelos.

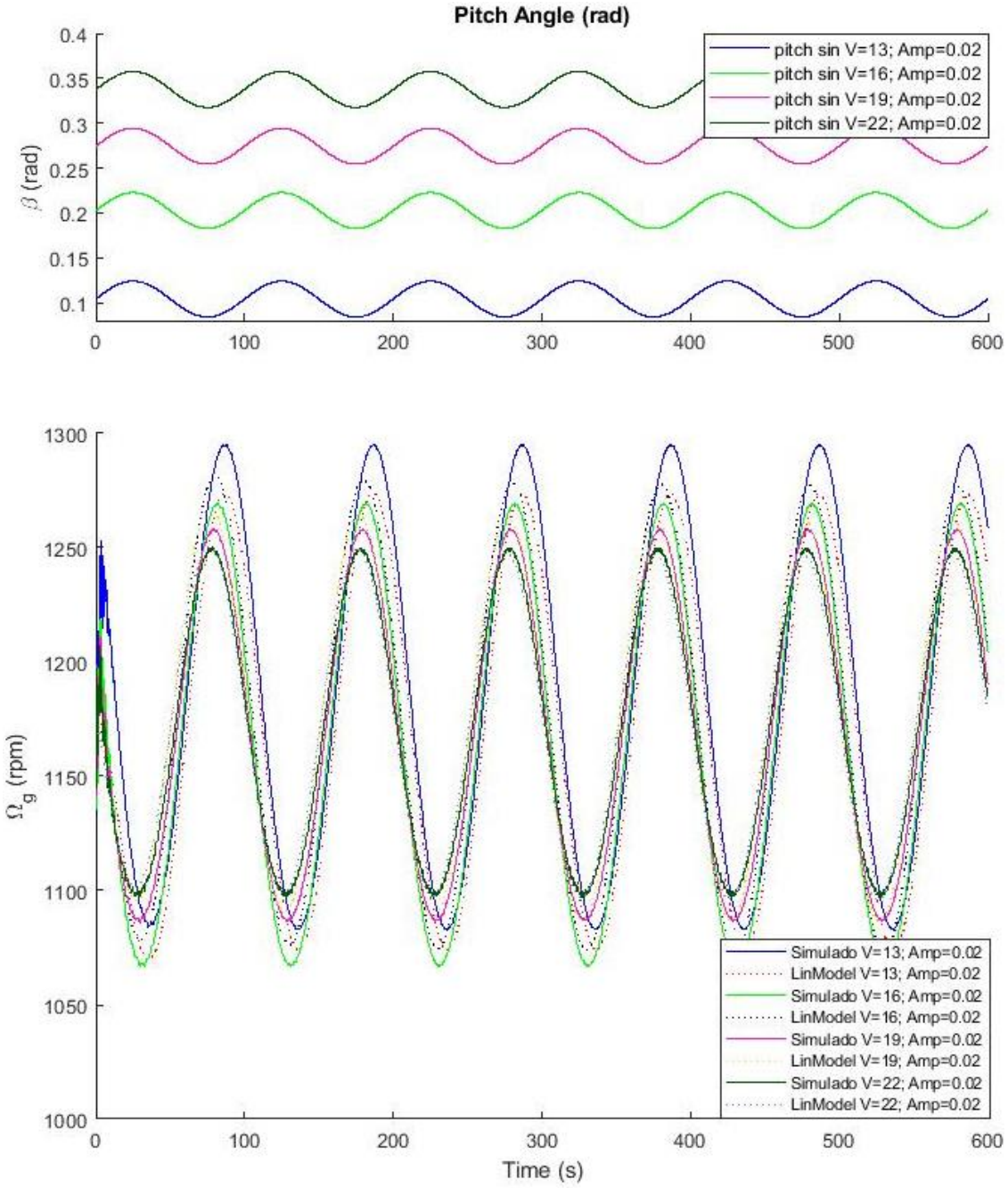

Figura 63. Seno de pitch. Ω_g

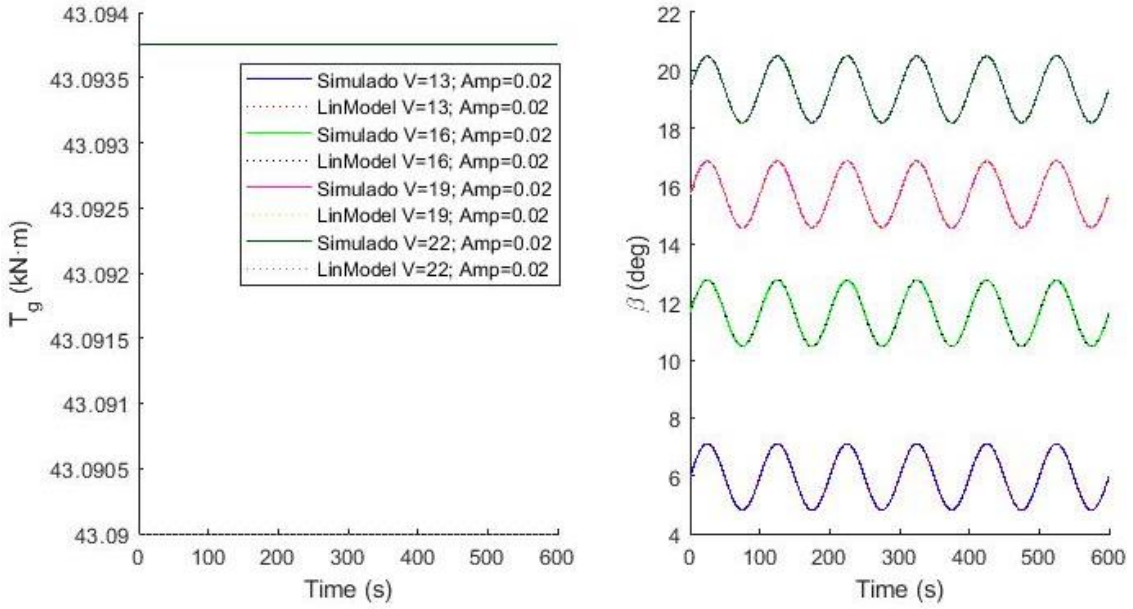

Figura 64. Seno de pitch. Tg y β

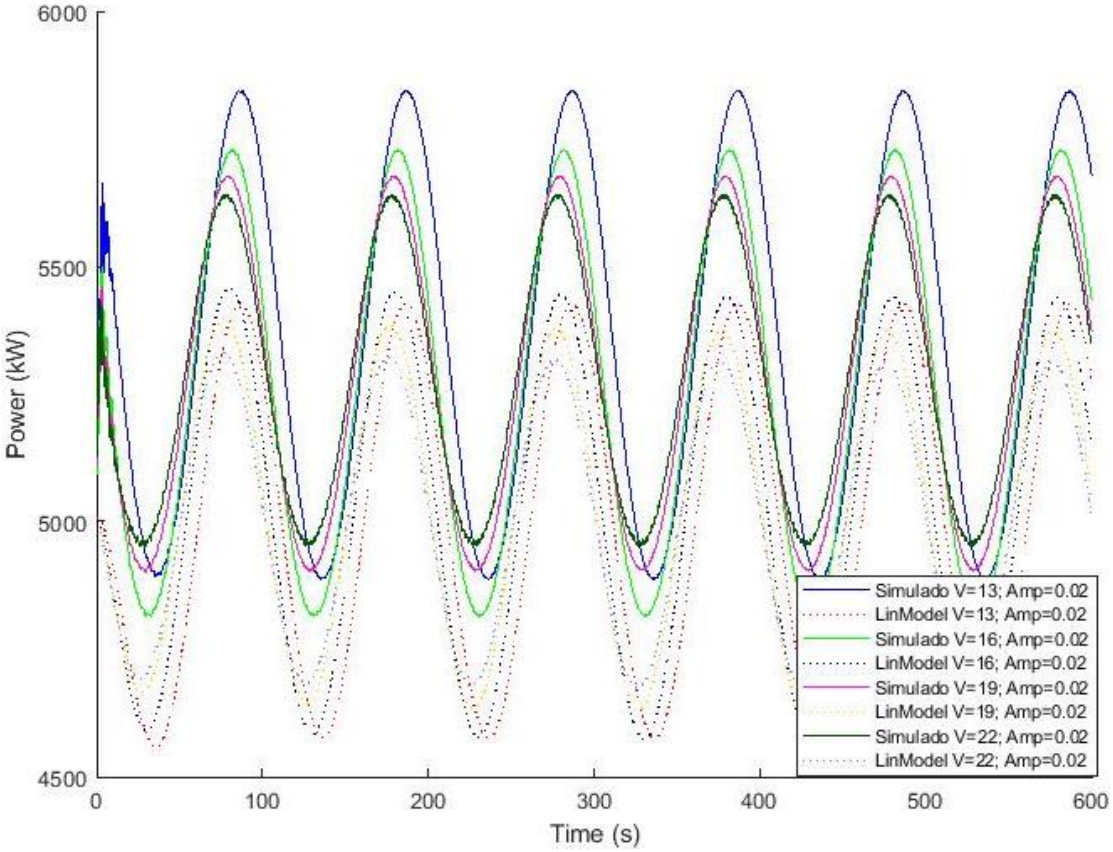

Figura 65. Seno de pitch. Potencia

Se introduce un **seno de par** de amplitud 1000 N·m cuando la máquina está trabajando a 4, 6, 8 y 10 m/s (Figura 66Figura 42). Cuando la máquina está trabajando a 4 m/s y 6 m/s, el modelo lineal se frena rápidamente y no alcanza el régimen estacionario. Este comportamiento es normal ya que un escalón de amplitud 1000 N·m ante un par de equilibrio de 3.5 kN·m inicial

correspondiente a un viento de 4 m/s y 11 kN·m a 6 m/s es porcentualmente mucho mayor que un par de equilibrio de 30 kN·m correspondiente a un viento de 10 m/s (Figura 67).

Al igual que en los anteriores casos, el punto de equilibrio entre el modelo lineal y el no lineal difieren, especialmente a 6, 8 y 10 m/s. Sin embargo la dinámica es similar en ambos modelos (Figura 66 y Figura 68). La diferencia en el punto de equilibrio se traduce en la variación del valor medio de la señal senoidal.

El pitch demandado se establece con valor nulo (Figura 67) debido a que los senos de par han sido introducidos en las zonas 1.5, 2 y 2.5 en las que hay control de par y no de pitch.

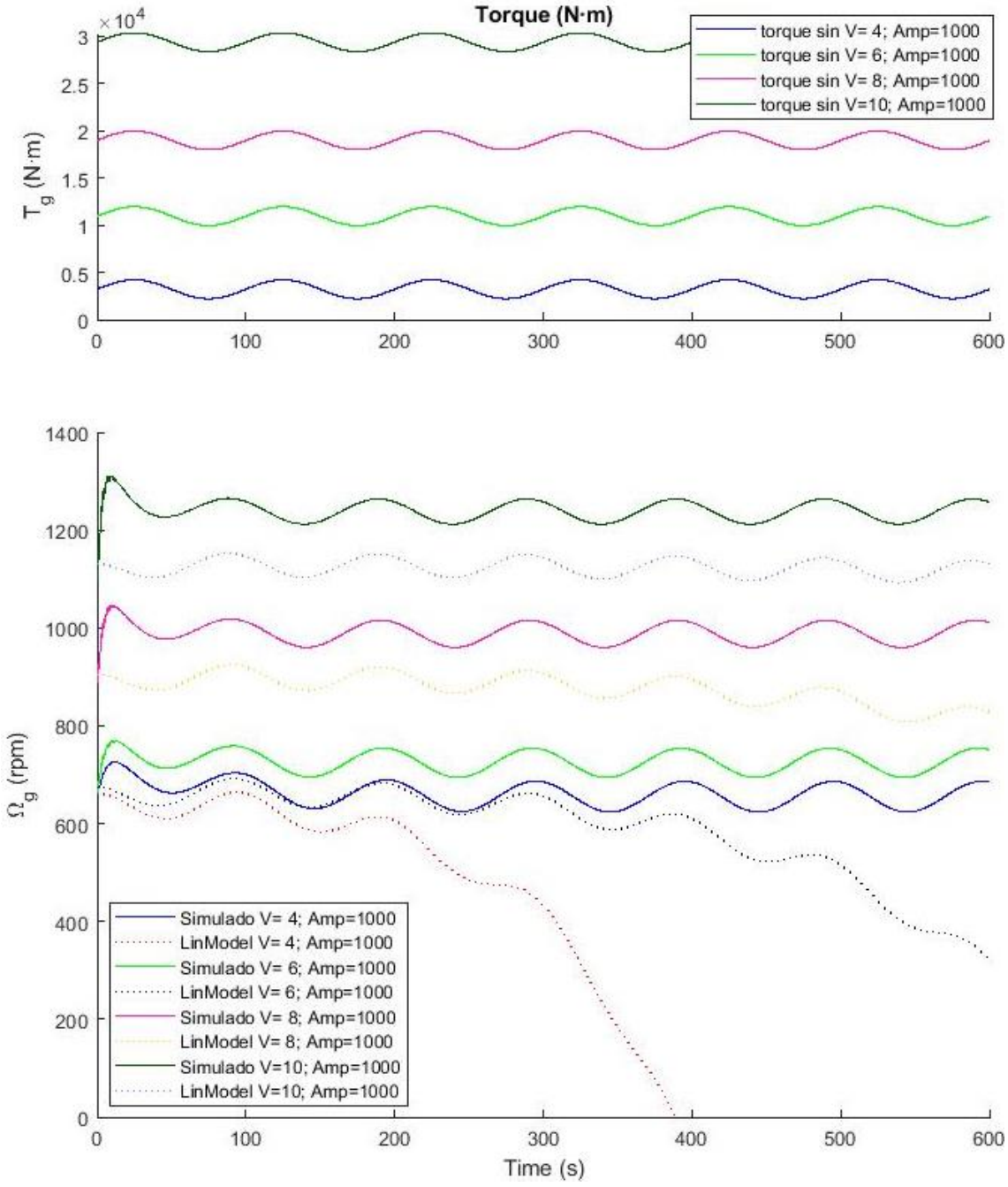

Figura 66. Seno de par. Ω_g

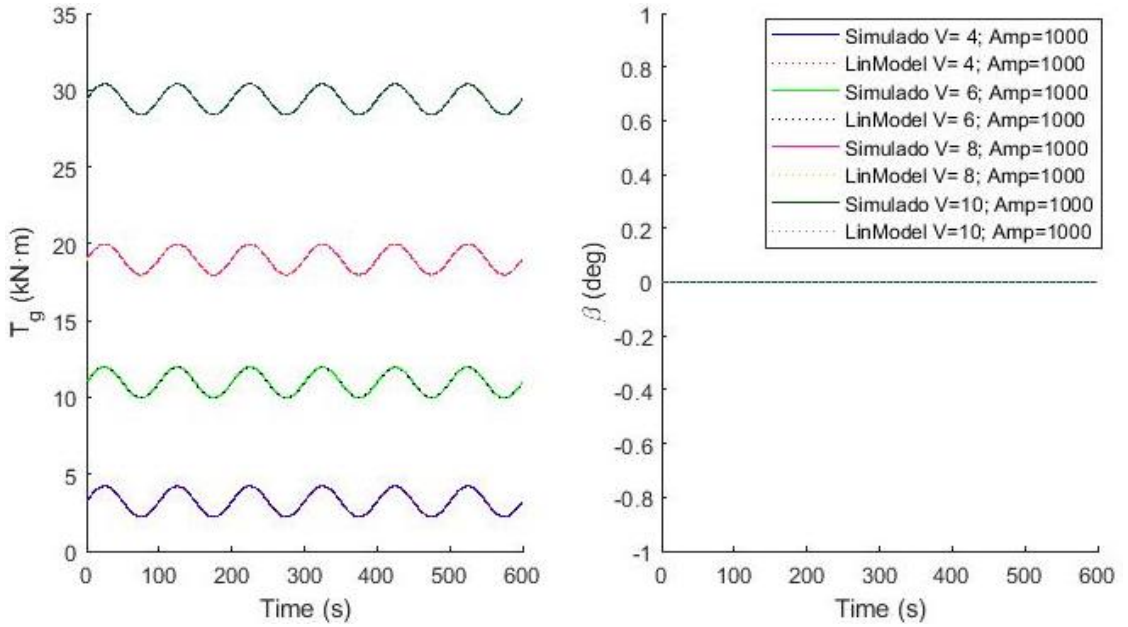

Figura 67. Seno de par. Tg y β

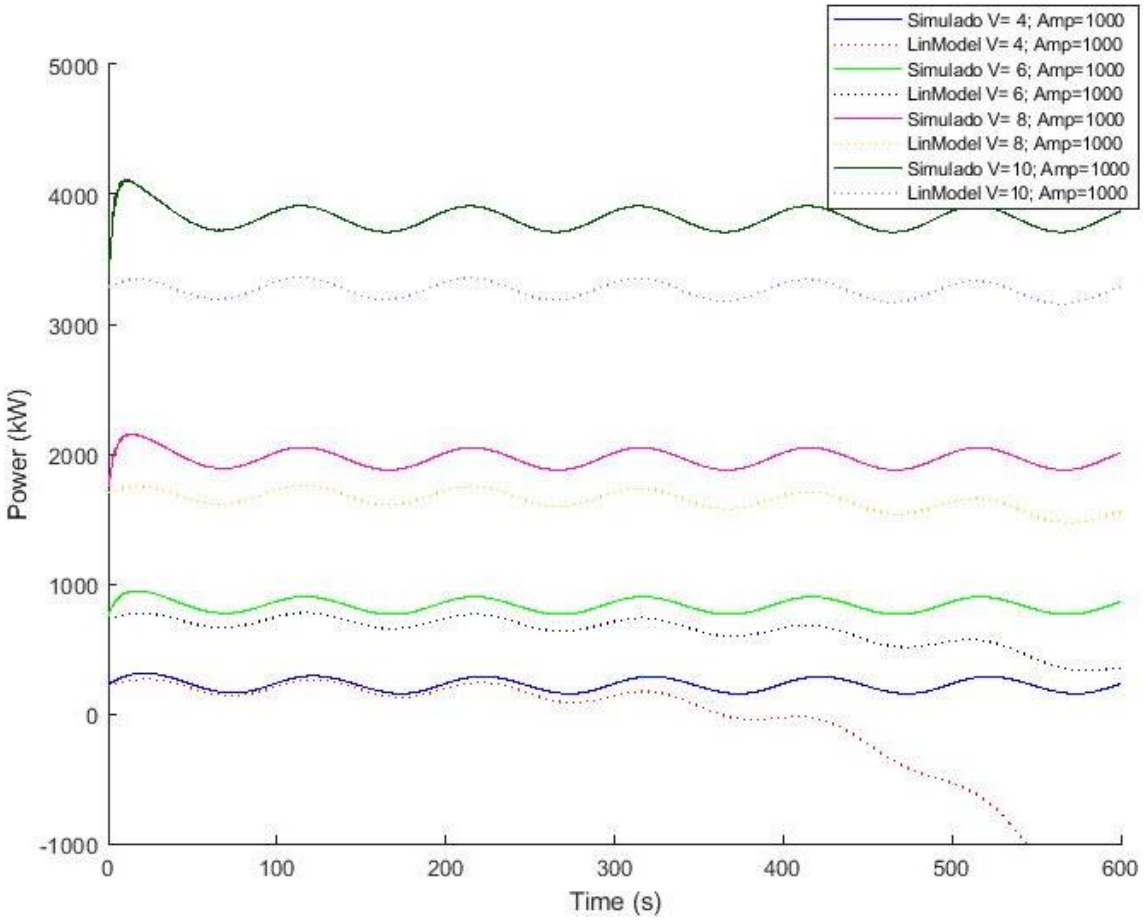

Figura 68. Seno de par. Potencia

Diagrama de Bode

Los diagramas de Bode obtenidos con el modelo lineal de FAST están incluidos en el apartado *Modelo lineal obtenido*. Sin embargo, un diagrama de Bode puede definirse como la representación gráfica de la respuesta en frecuencia del sistema y, por lo tanto, la entrada senoidal a una determinada frecuencia del sistema no lineal se traduce en un punto del diagrama de Bode.

Tal y como se ha obtenido en el anterior apartado, al introducir una entrada senoidal al sistema no lineal de FAST, la salida se corresponden con un seno desfasado y de diferente ganancia. Si se simula el modelo con entradas sinusoidales a diferentes frecuencias, se puede obtener el diagrama de Bode del modelo no lineal de FAST.

Para ello se ha desarrollado un código que simula y almacena la respuesta del sistema global de FAST ante entradas senoidales a diferentes frecuencias y diferentes velocidades de viento (ANEXO E: Validación Diagrama de Bode). Para cada frecuencia simulada se almacena el desfase entre la entrada y la salida, así como la relación de ganancia (Figura 69). Una vez simulado para un conjunto de frecuencias, se obtiene el diagrama de Bode correspondiente a la velocidad de viento seleccionada y para la entrada escogida.

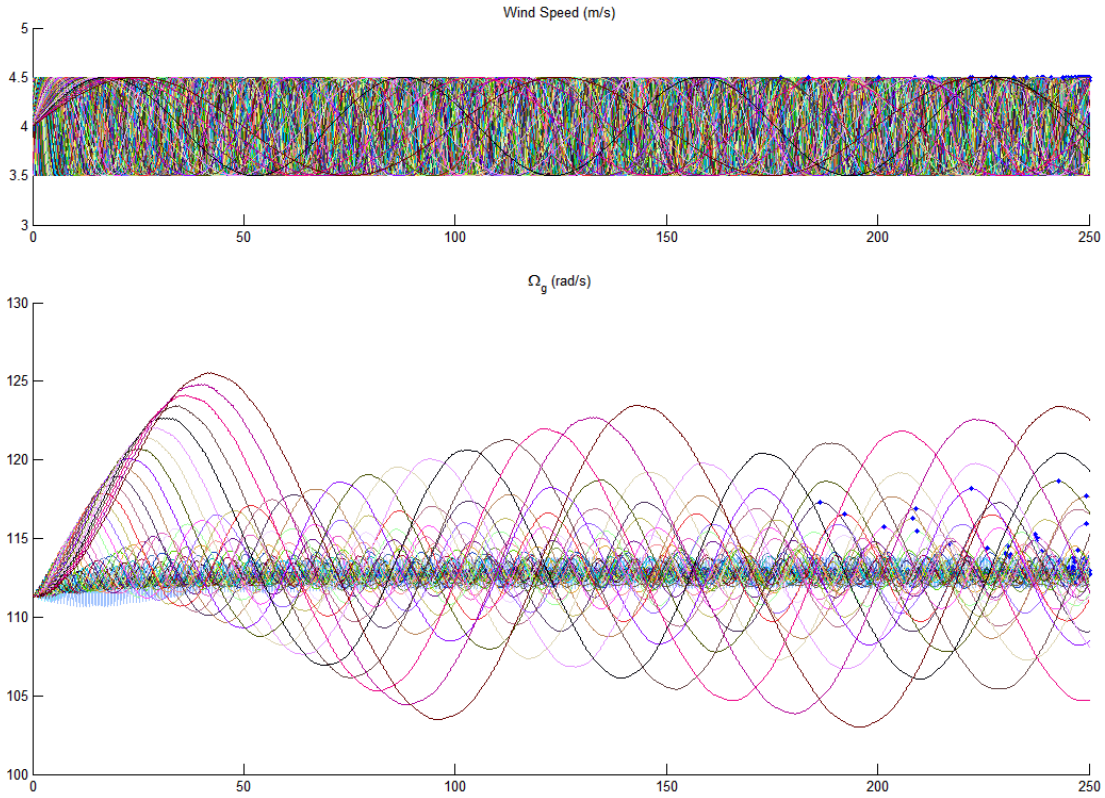

Figura 69. Simulación a varias frecuencias de una entrada de viento senoidal a 4 m/s

A pesar de poder obtener los diagramas de Bode del sistema no lineal, siendo supuestamente estos más fieles que los del modelo lineal, no se emplean para el diseño del control ya que el coste computacional de obtener un diagrama de Bode de muchos puntos es muy alto.

Las siguientes figuras representan el diagrama de Bode tanto del modelo lineal como el modelo no lineal calculado para las entradas viento, pitch y par, y salidas velocidad de giro del rotor, velocidad de giro del generador y potencia. También se obtienen los diagramas de Bode pitch demandado – pitch real (Figura 76) y par demandado – par real (Figura 80).

Los resultados obtenidos para la **entrada viento**, tanto para la salida de velocidad de giro del rotor (Figura 70) como la velocidad de giro del generador (Figura 71) son semejantes. Las mayores diferencias se encuentran en el modelo lineal a 4 m/s, pero teniendo en cuenta que para realizar dicho diagrama de Bode se han empleado senos de la misma amplitud independientemente de la velocidad de viento media, es normal que a 4 m/s ambos modelos difieran (proporcionalidad).

Para la salida de potencia (Figura 72) también se obtienen resultados simulares al modelo lineal. Si se aumentase el número de muestras a partir de 0.5 Hz, así como el tiempo de simulación ante la entrada senoidal, se podrían obtener resultados más fiables ya que el sistema estaría más cerca de alcanzar la estabilidad y con ello obtener la fase y ganancia real. Sin embargo, debido al alto coste computacional, solo se han tomado 100 muestras.

Los resultados obtenidos para la **entrada pitch** son similares para la salida de velocidad de giro del rotor (Figura 73), velocidad de giro del generador (Figura 74) y potencia (Figura 75). Las mayores diferencias se encuentran en la fase, pero podría solucionarse aumentando el número de muestras (datos a más frecuencias) y aumentando el tiempo de simulación (sistema más próximo a la estabilidad). En la Figura 76 se puede observar el fenómeno descrito previamente del actuador ideal de pitch (30): ganancia constante a 35.15 dB y fase 0.

Los resultados obtenidos para la **entrada par** son similares para la salida de velocidad de giro del rotor (Figura 77), velocidad de giro del generador (Figura 78) y potencia (Figura 79). Cuanta más alta la frecuencia, mejores resultados se han obtenido. En el diagrama de Bode de potencia (Figura 79) hay un pequeño offset en para cada una de las velocidades de viento; sin embargo, el modelo no lineal se asemeja más al modelo lineal para la entrada par que para las anteriores entradas de viento y pitch.

Al igual que para la entrada pitch, el diagrama de Bode de demanda de par a par real, se obtiene una ganancia constante e igual a -60 con una fase constante de 0. Esto se debe a que el actuador de par se ha considerado ideal (29).

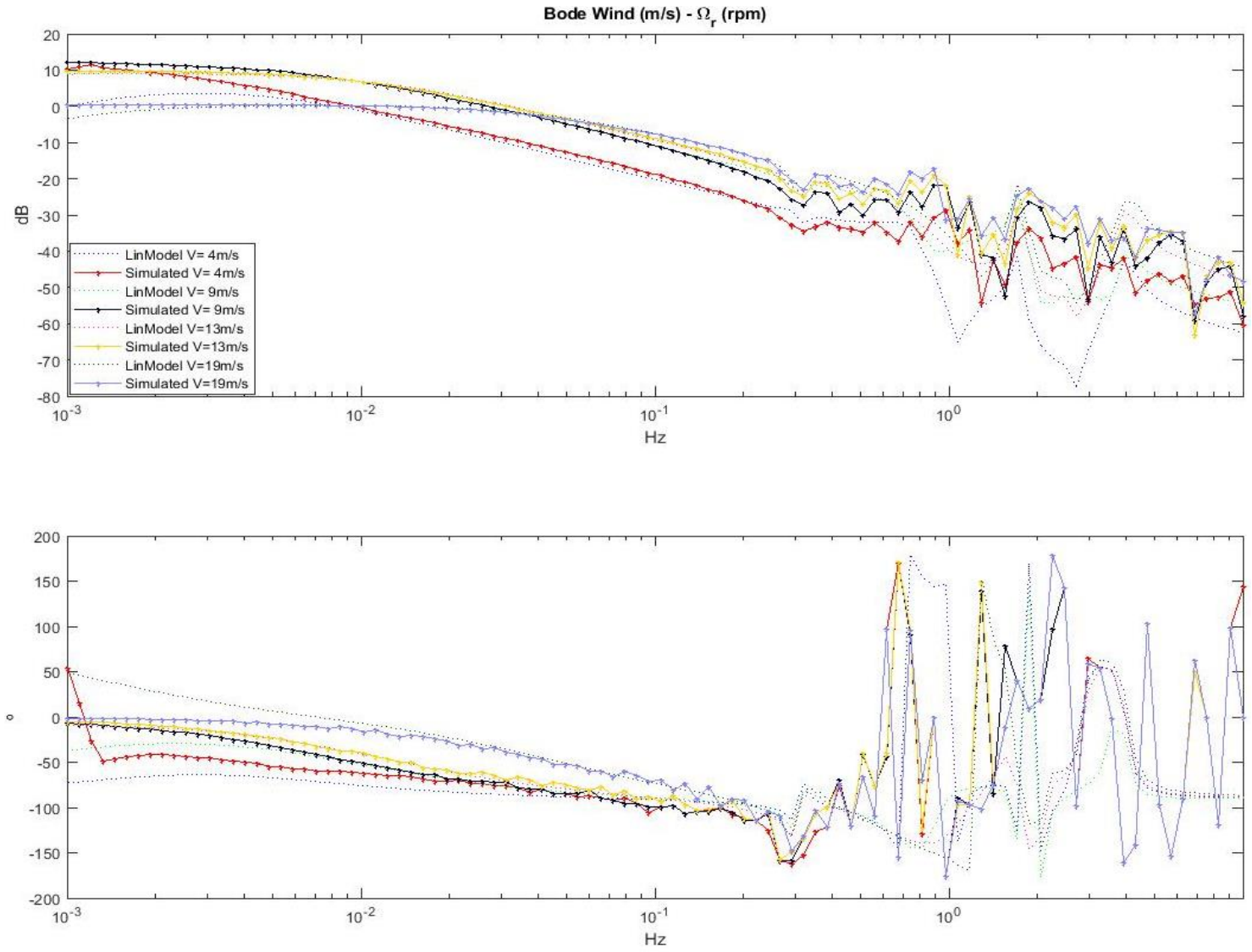

Figura 70. Diagrama de Bode FAST simulado. Viento - Velocidad de giro del rotor

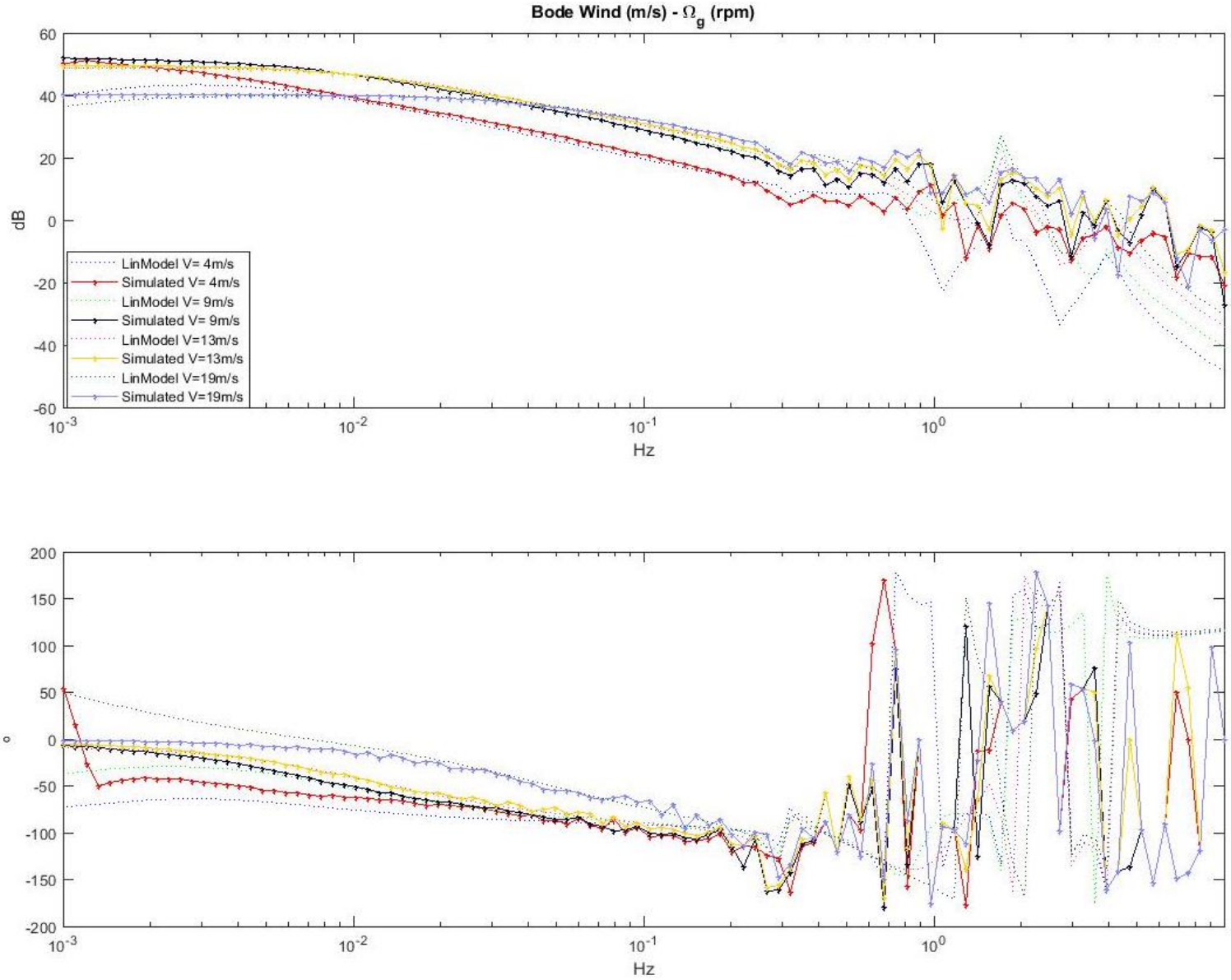

Figura 71. Diagrama de Bode FAST simulado. Viento - Velocidad de giro del generador

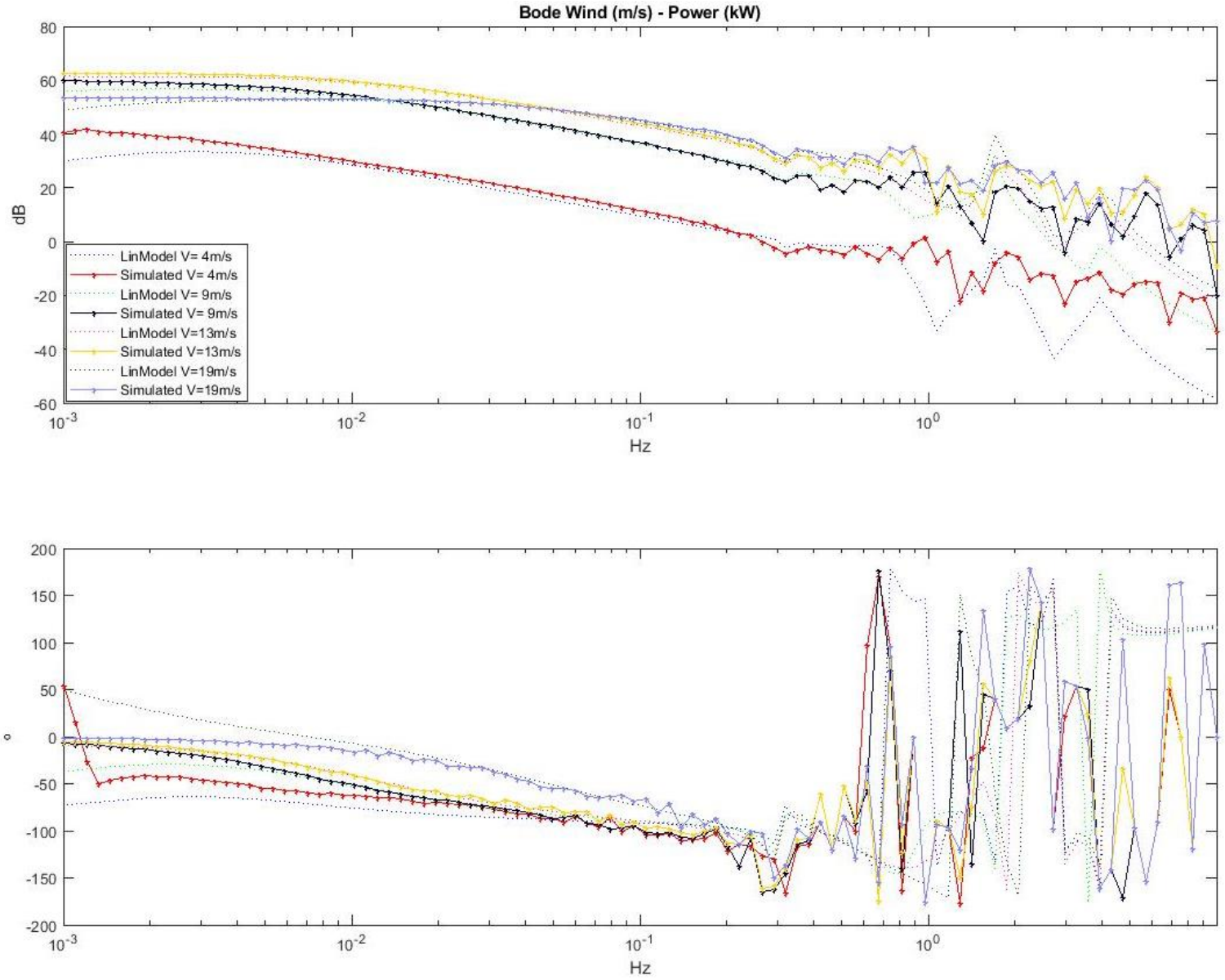

Figura 72. Diagrama de Bode FAST simulado. Viento - Potencia

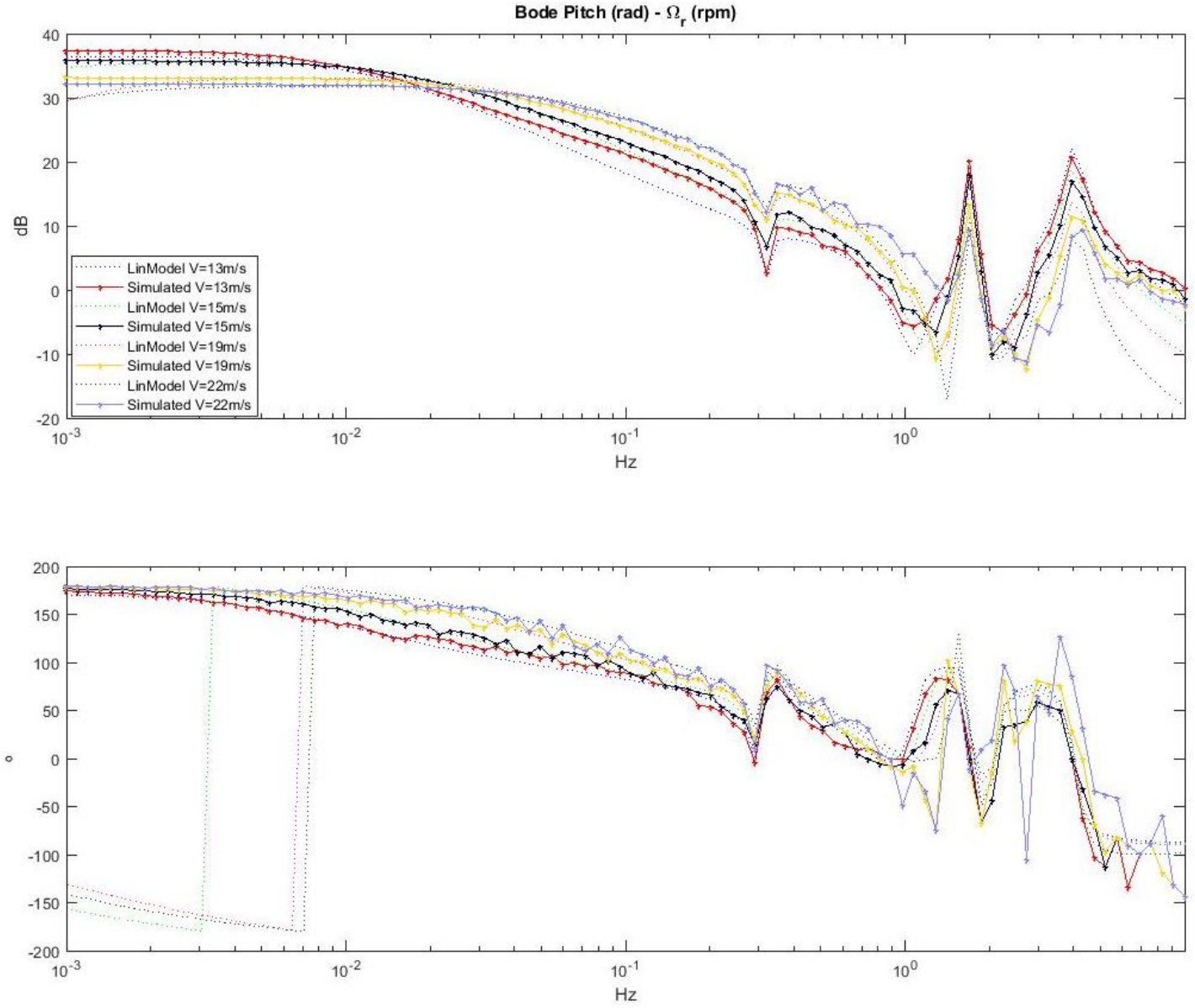

Figura 73. Diagrama de Bode FAST simulado. Pitch - Velocidad de giro del rotor

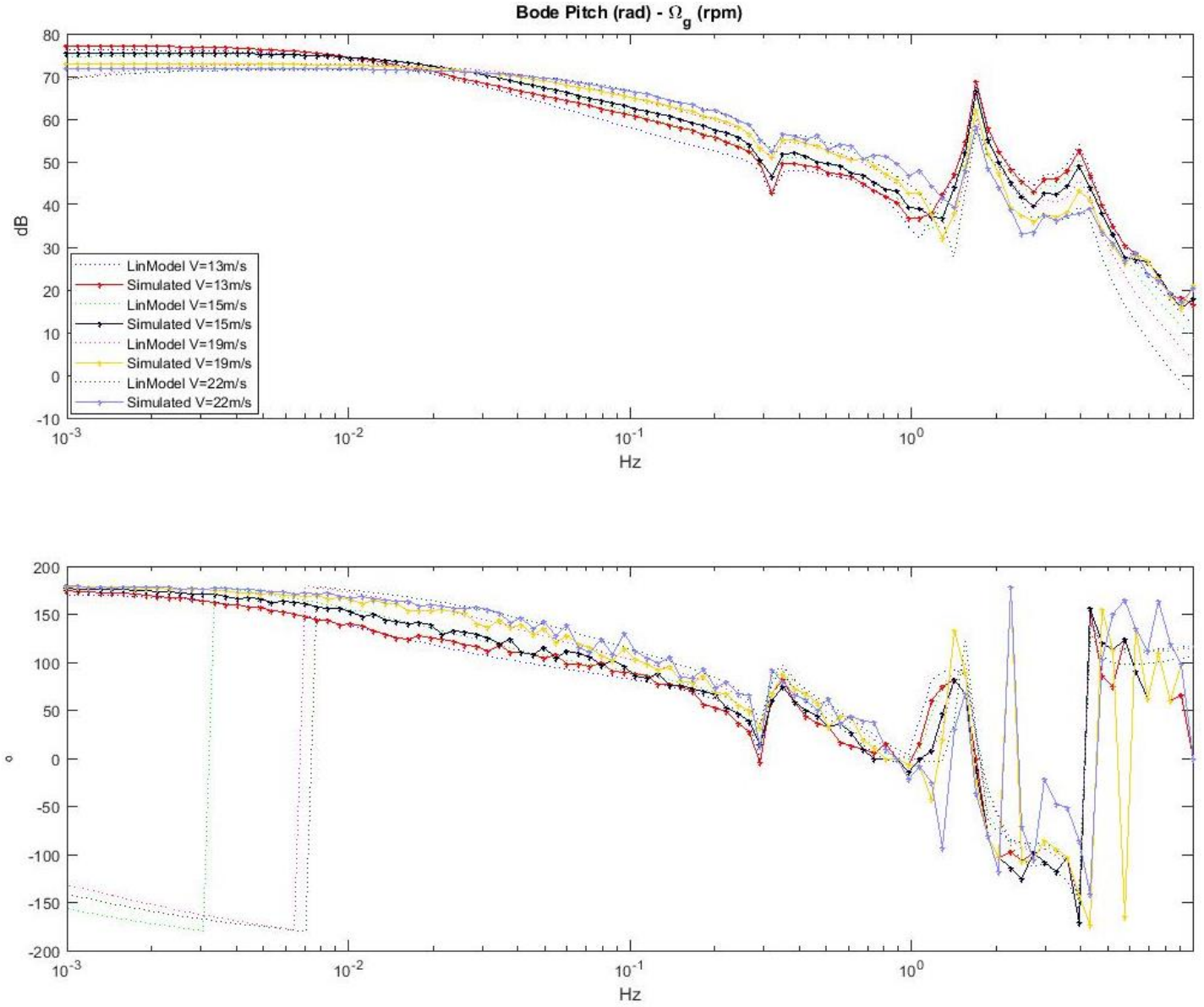

Figura 74. Diagrama de Bode FAST simulado. Pitch - Velocidad de giro del generador

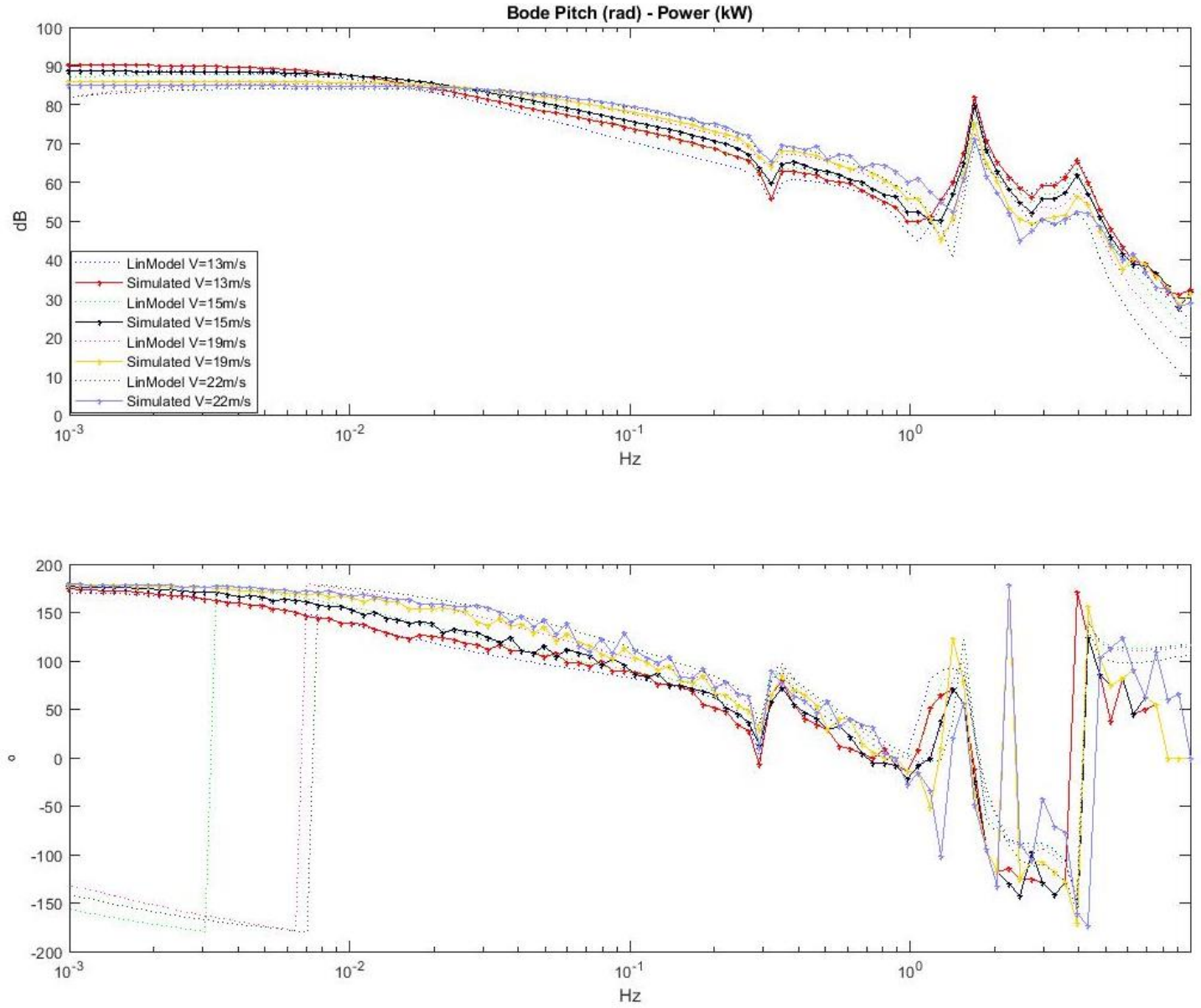

Figura 75. Diagrama de Bode FAST simulado. Pitch - Potencia

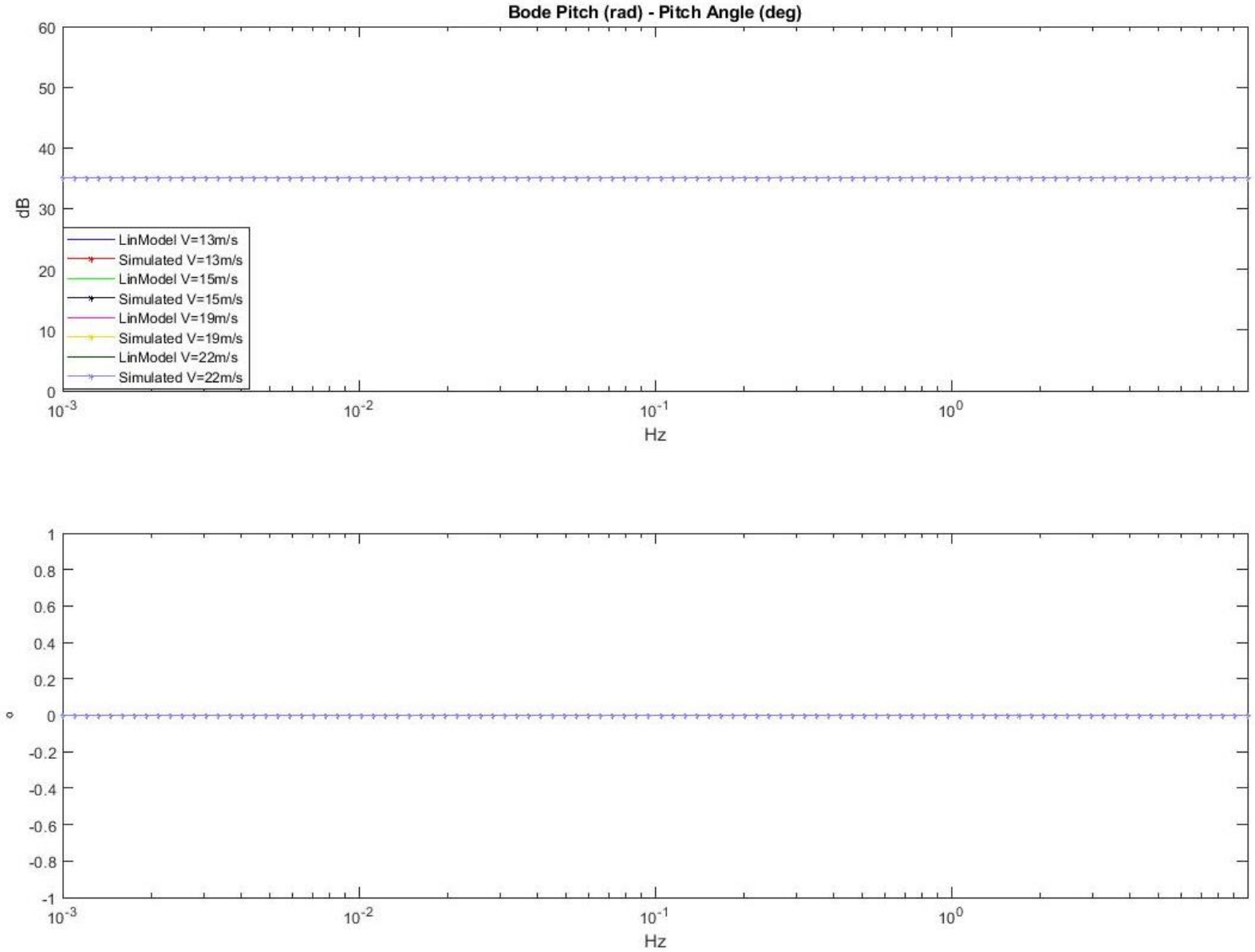

Figura 76. Diagrama de Bode FAST simulado. Pitch - Pitch

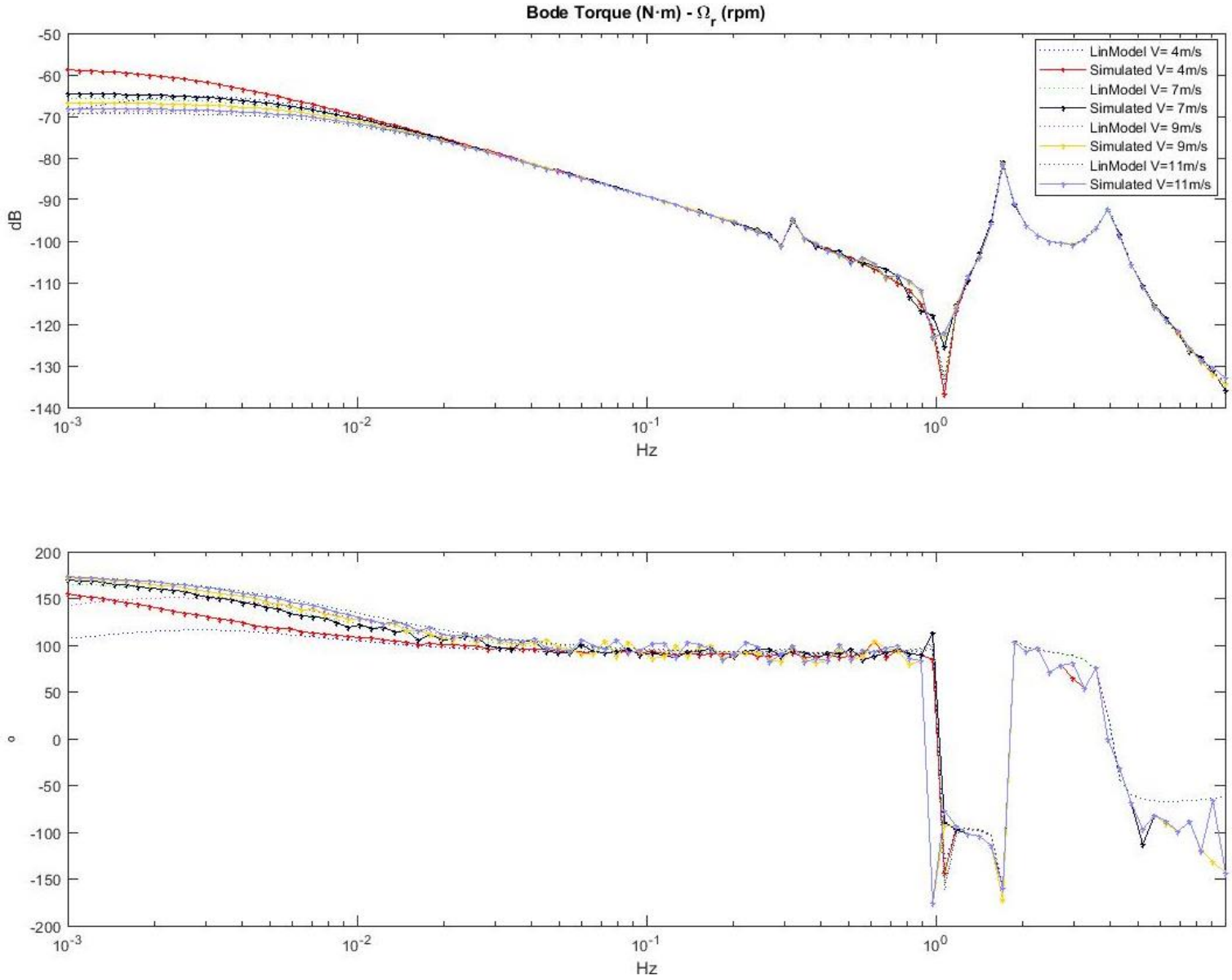

Figura 77. Diagrama de Bode FAST simulado. Par - Velocidad de giro del rotor

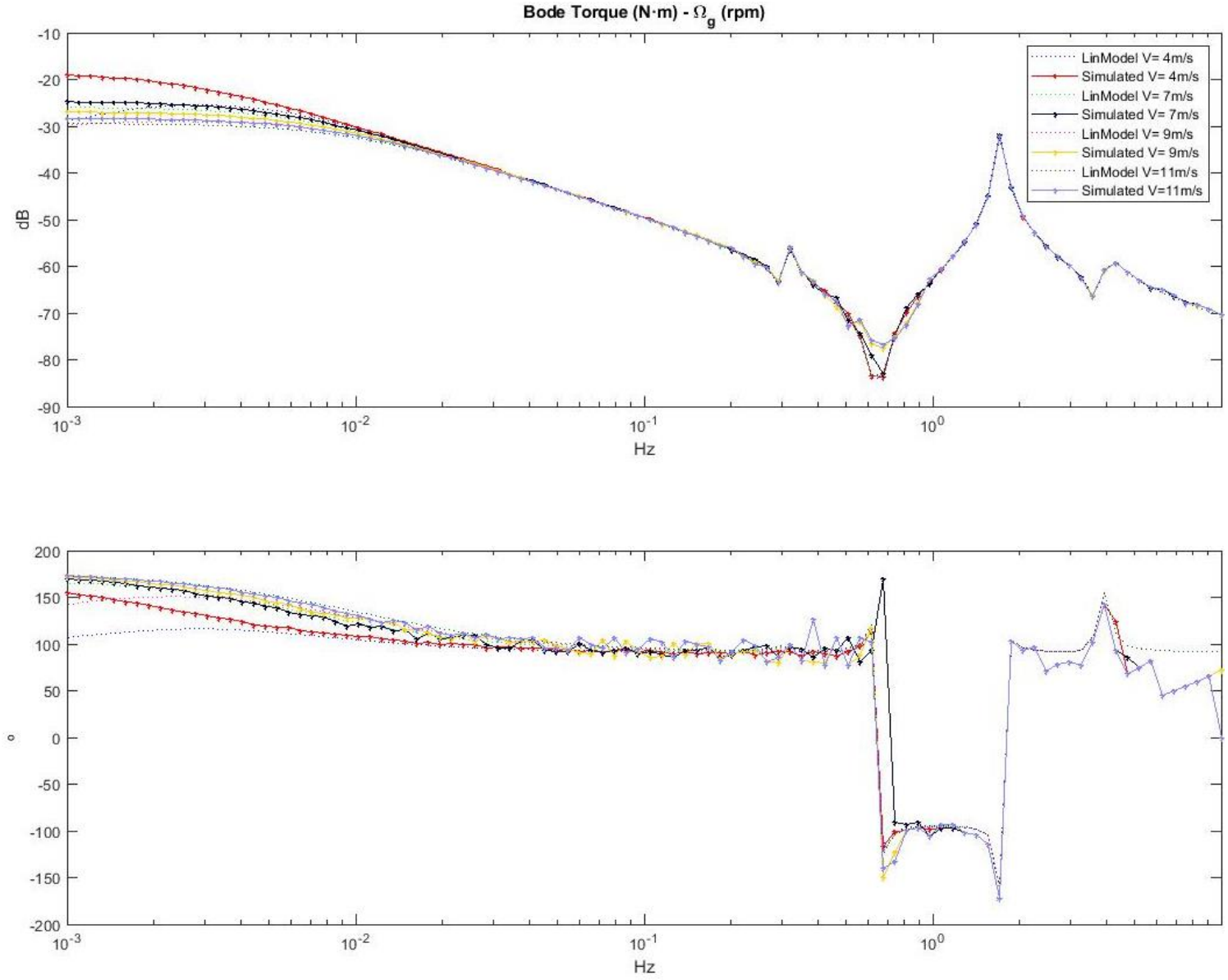

Figura 78. Diagrama de Bode FAST simulado. Par - Velocidad de giro del generador

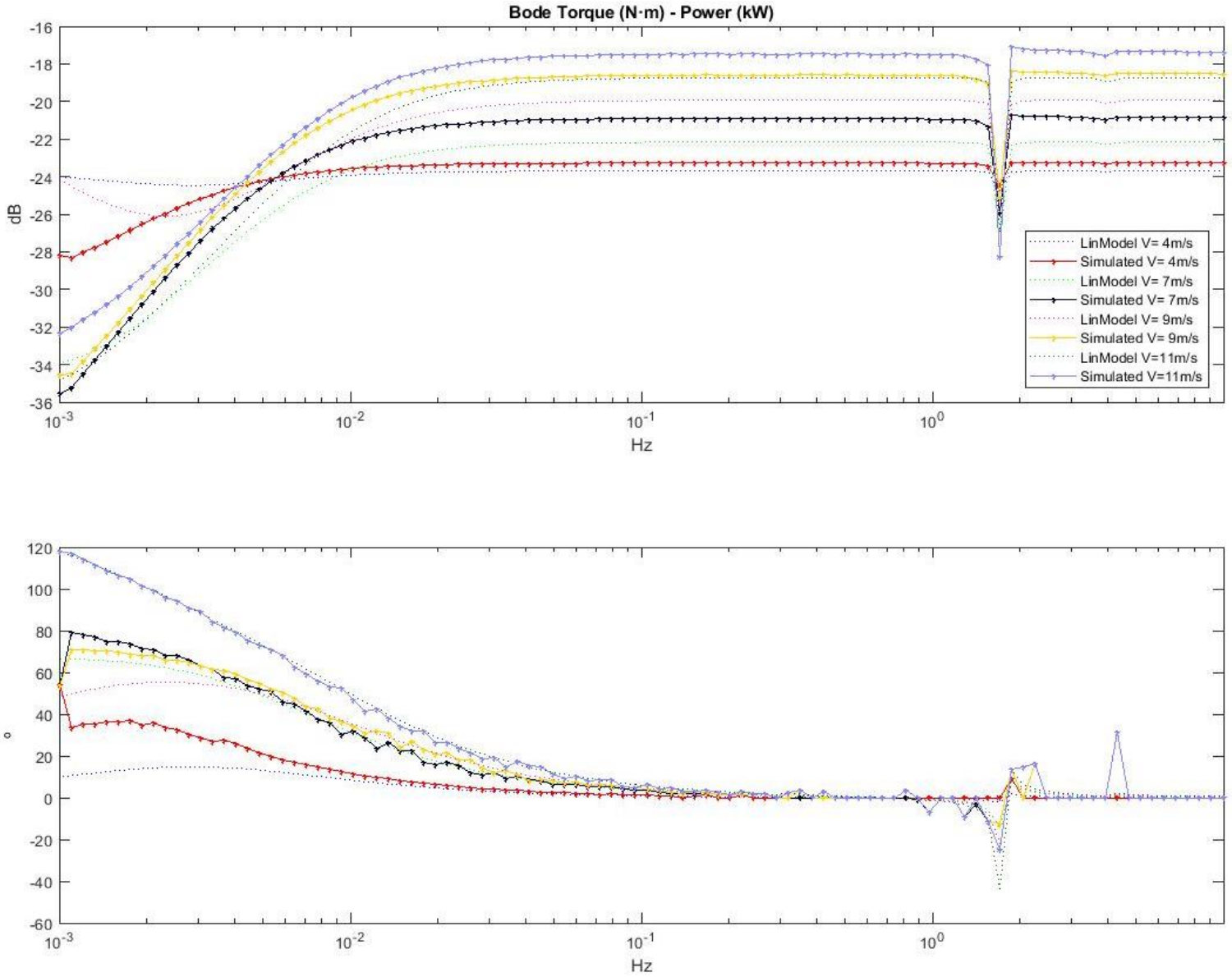

Figura 79. Diagrama de Bode FAST simulado. Par – Potencia

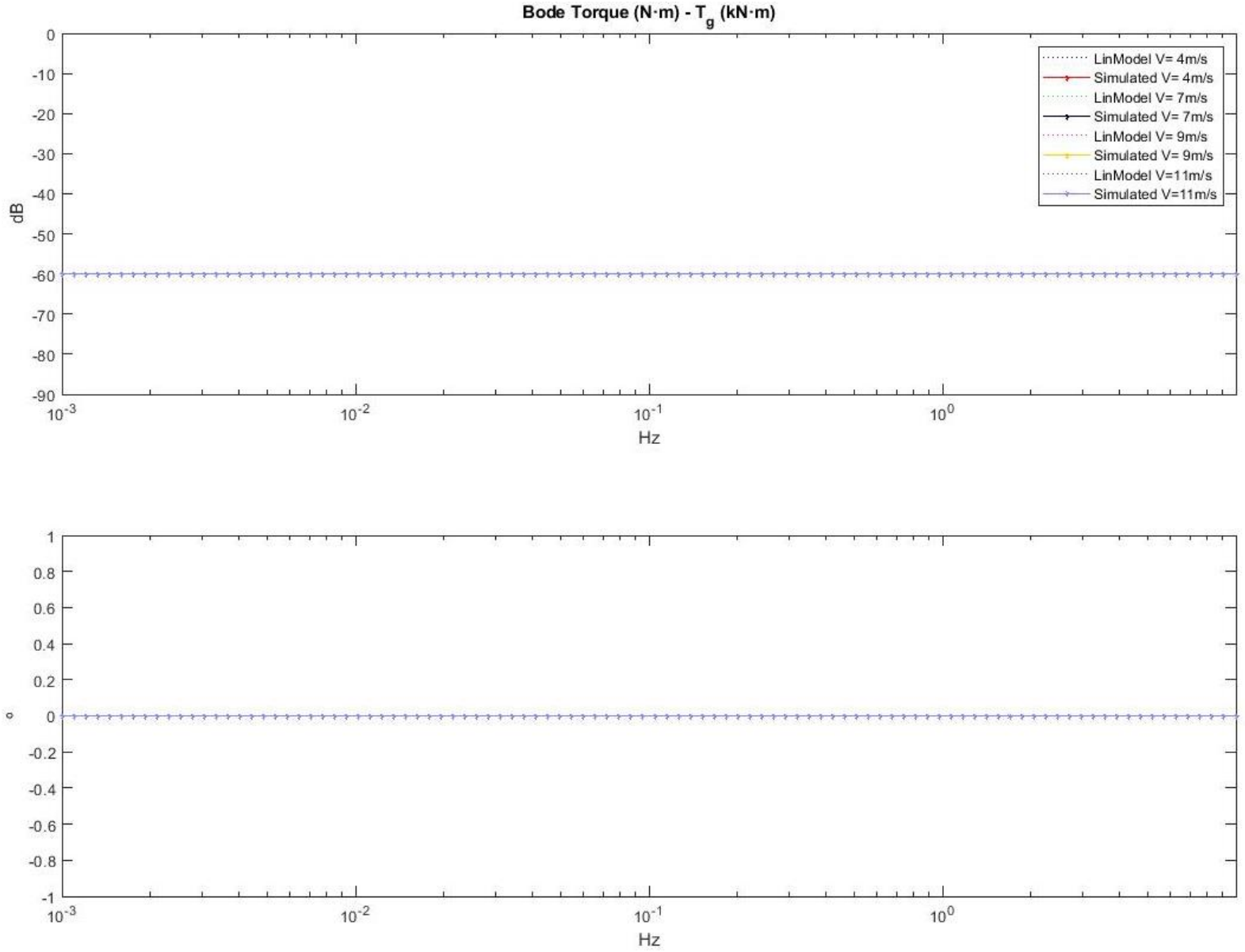

Figura 80. Diagrama de Bode FAST simulado. Par - Par

Lazo cerrado

Para poder validar la linealización del sistema, no sólo se compara el sistema en lazo abierto, sino que también se estudia su comportamiento en lazo cerrado. El cierre de lazos implica la definición de bloques de control para cada una de las variables del sistema.

Debido a que el comportamiento del aerogenerador se ve fuertemente influenciado por la velocidad de viento, así como por la región de operación, el control se divide principalmente en:

- Control de par. Según la zona de operación, la demanda de par que se introduce en el sistema se obtiene con un controlador específico:
    - Tcntrl1. El controlador debe tener una parte integral, ya que en la zona 1.5, la máquina debe seguir la primera vertical, y por lo tanto, mantener la velocidad de giro del generador igual a la mínima ($\Omega_{g,min}$) mediante la demanda de par.
    - Kopt. En la zona 2, el coeficiente de potencia de la máquina se maximiza mediante el empleo de la constante K_{opt} (ecuaciones (22) y (23)).
    - Tcntrl2. El controlador debe tener una parte integral, ya que en la zona 2.5, la máquina debe seguir la segunda vertical, y por lo tanto, mantener la velocidad de giro del generador igual a la nominal ($\Omega_{g,nom}$) mediante la demanda de par.
- Control de pitch – Scntrl. En la zona 3 se impone una demanda de par constante e igual a la nominal, mientras que se mantiene la velocidad de giro del generador ($\Omega_{g,nom}$) mediante la variación de ángulo de paso de pala. Cuanto mayor pitch, menor potencia del viento es aprovechada, y por lo tanto, menor coeficiente de potencia. Sin embargo, si no se limitase la potencia, la máquina sufriría daños.

Además de los controles de par y pitch debidas a las zonas de operación del aerogenerador, se definen otros dos controladores:

- ATD (Active Tower Damping). A partir de la medida de aceleración de la góndola, actúa sobre la demanda de pitch para conseguir un empuje contrario al viento y reducir dicha aceleración.
- DTD (Drive Train Damping). A partir de la velocidad del generador, actúa sobre la demanda de par para reducir las vibraciones debidas a la torsión del eje de potencia del aerogenerador.

Tanto el ATD como el DTD son controladores que deben funcionar en todas las zonas de operación, ya que en otro caso el aerogenerador podría entrar en estado de emergencia o directamente romperse.

El diseño de todos los controladores que se van a emplear, se realiza a partir de los diagramas de Bode a las velocidades de diseño del aerogenerador o planta. El diseño de cada controlador y su estructura escogida se estudia y analiza en el documento Complemento al Trabajo Fin de Máster [7].

A diferencia de otros esquemas que se emplean en la simulación de aerogeneradores, los controladores escogidos siempre van a estar activos. Cuando el aerogenerador está trabajando en una zona en la cual el controlador se supone que no debe actuar, sus parámetros tienen valor nulo. De esta forma, se puede eliminar el *'switch'* entre zonas que se suele emplear.

Además, la estructura escogida permite el empleo de un controlador para cada velocidad de viento, suavizando la transición entre zonas y optimizando la máquina. De manera gráfica podría representarse como:

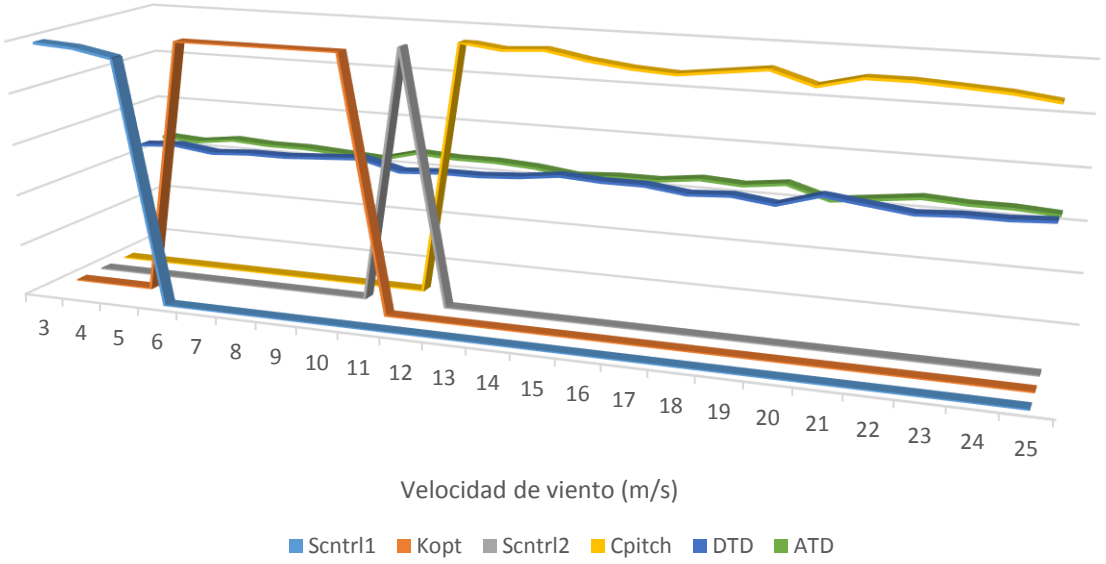

Figura 81. Transición controladores

En la implementación del cierre de lazos con los controladores escogidos para el modelo lineal, se emplea la función *connect* de MATLAB. Dicha función se encarga de conectar bloques previamente definidos a través de los nombres de entrada y de salida. Se deben definir además de los bloques de los controladores, los bloques suma Pitch, Nacx, Tg, Ω_g y Ω_{error}.

En cuanto al cierre de lazos del modelo no lineal o sistema real, se completa la estructura de Simulink con la función interfaz de FAST empleada previamente en lazo abierto. La estructura completa, así como su máquina de estados, es explicada y definida en el Complemento al Trabajo Fin de Máster [7]. Para acceder a los valores de cada controlador, se emplean bloques *Lookup Table* (Figura 82).

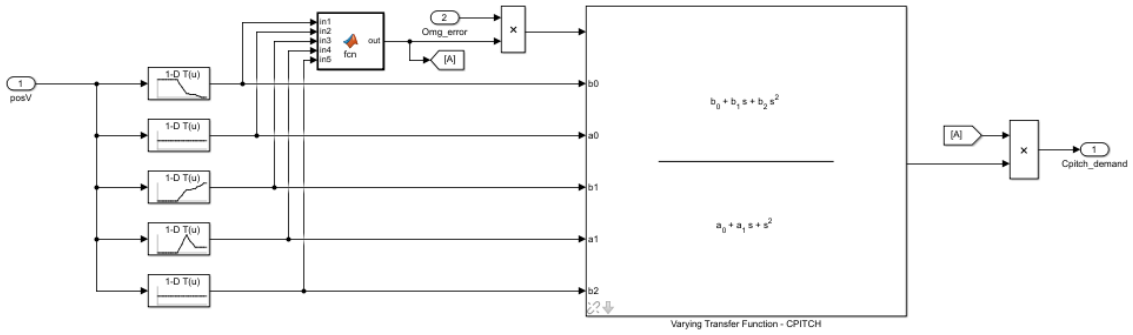

Figura 82. Implementación controladores en Simulink

Tanto para el modelo lineal como el modelo simulado con FAST, la estructura de bloques escogida se corresponde con la representada en la Figura 83.

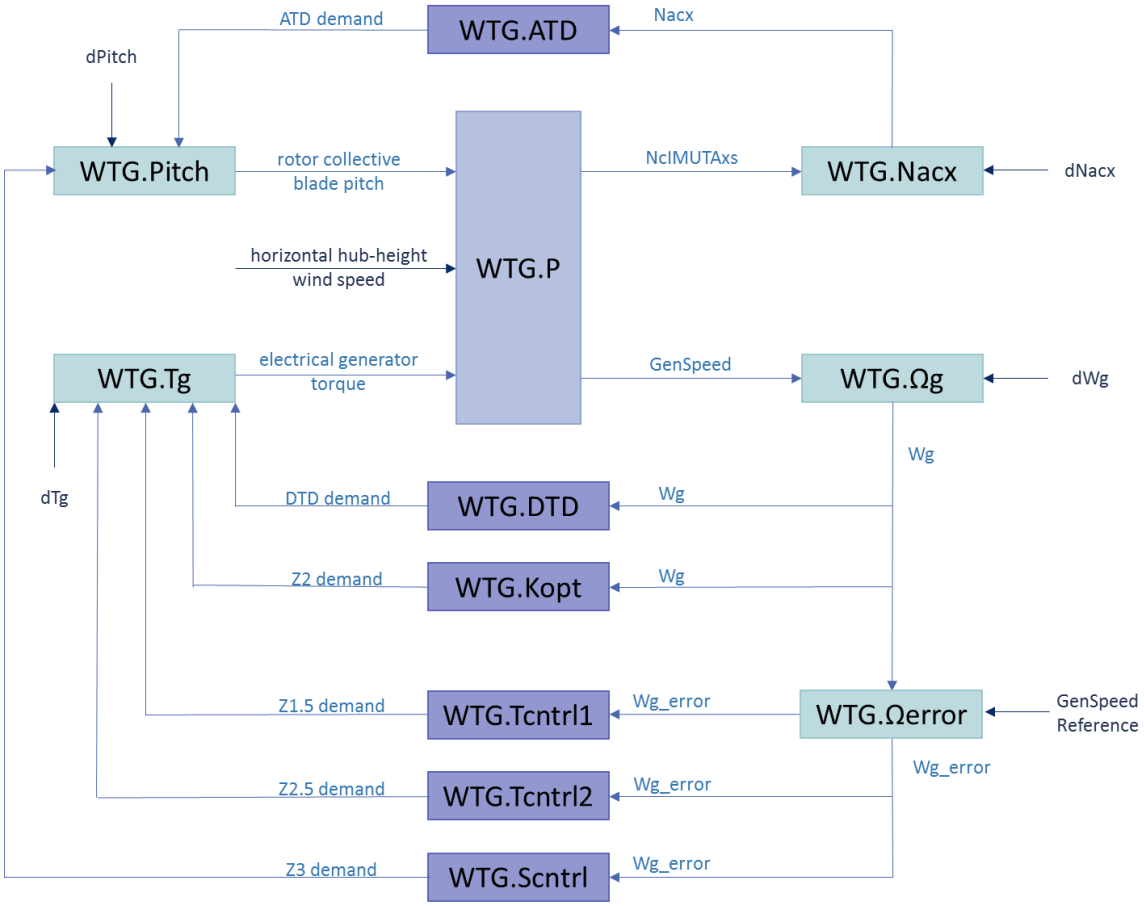

Figura 83. Estructura de control

Una vez cerrados los lazos del modelo lineal y la estructura de Simulink con FAST, se comparan los resultados obtenidos tanto para entrada escalón, como rampa y senoidal. Finalmente se hace una simulación rampa ascendente y otra descendente de todas las zonas de operación. En la siguiente tabla se recogen todas las validaciones que se llevan a cabo.

Tipo de entrada	Entrada	Amplitud	Velocidades de viento
Escalón $t_{step} = 200s$ $t_{end} = 400s$	Viento	1 m/s	4 7 11 16 22
	Perturbación de Pitch	0.02 rad	13 16 19 22
	Perturbación de Par	1000 N	4 6 8 10
Rampa $t_{end} = 600s$	Viento	3 m/s	4 7 11 16 22
	Perturbación de Pitch	0.1 rad	13 16 19 22
	Perturbación de Par	2000 N	4 6 8 10
Senoidal $f = 0.01$ $t_{end} = 600s$	Viento	1 m/s	4 7 9 12 16 22
	Perturbación de Pitch	0.02 rad	13 16 19 22
	Perturbación de Par	1000 N	4 6 8 10
Completo $t_{end} = 1000s$ $t_{delay} = 50s$	Viento	+22 m/s	3
	Viento	-22 m/s	25

Tabla 11. Validaciones lazo cerrado

Escalón

Se introduce un **escalón de viento** de amplitud 1 m/s cuando la máquina está trabajando en lazo cerrado a 4, 7, 11, 16 y 22 m/s (Figura 84). El comportamiento de ambos modelos es similar salvo a 7 m/s que el modelo lineal parte de otro punto de equilibrio aunque la dinámica es la misma (Figura 84).

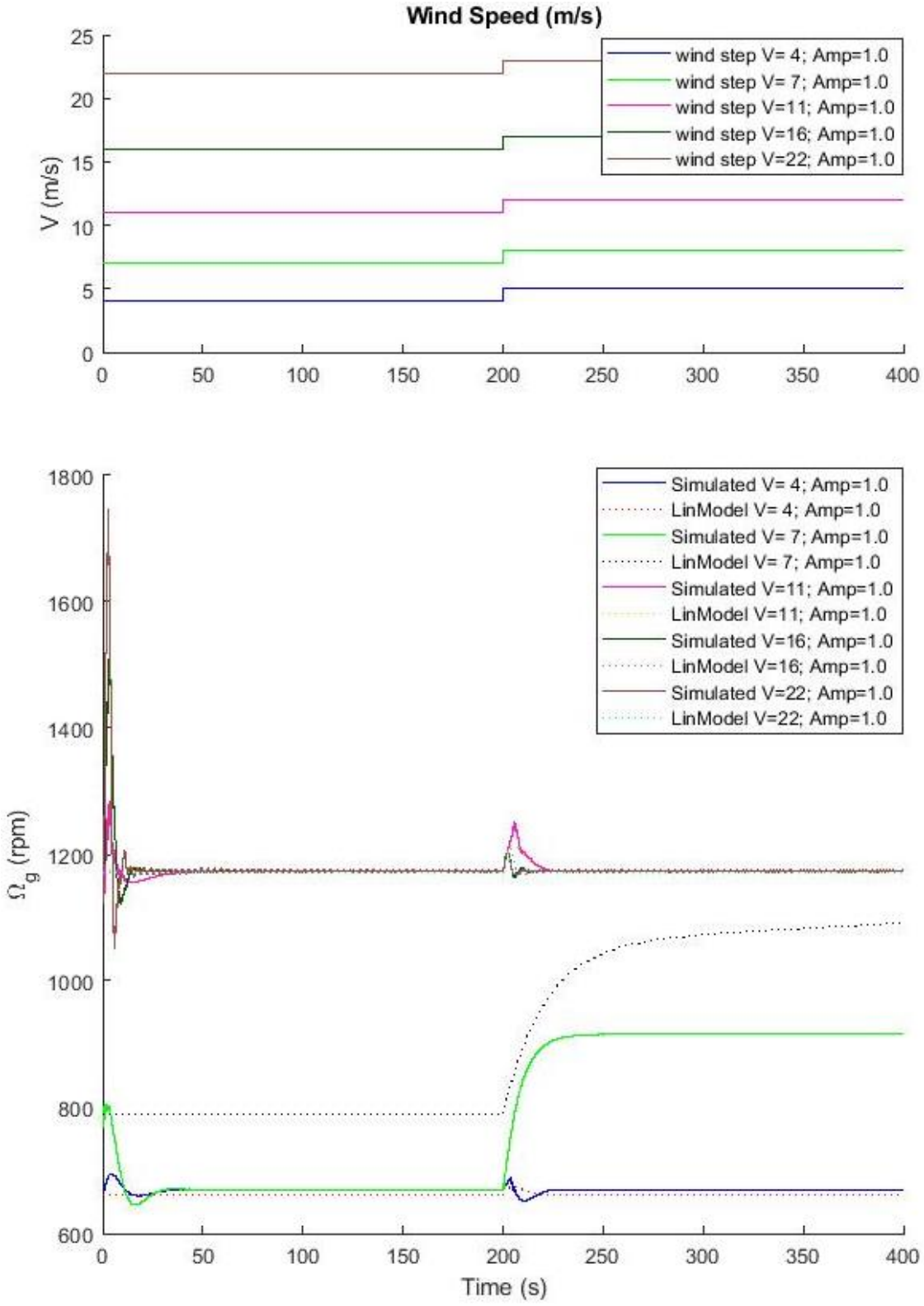

Figura 84. Escalón de viento en lazo cerrado. Ω_g

Al trabajar el lazo cerrado, el par y el pitch son calculados por el control diseñado. En la Figura 85 se observa que el modelo lineal a 11 m/s parte de zona de control de par, pero al emplear el mismo modelo lineal antes y después del escalón, tras el escalón sigue haciendo control de par cuando debería hacer control de pitch. Este resultado se corresponde con lo esperado ya que el modelo lineal no distingue entre zonas, sino que emplea el mismo modelo de la turbina para toda la simulación.

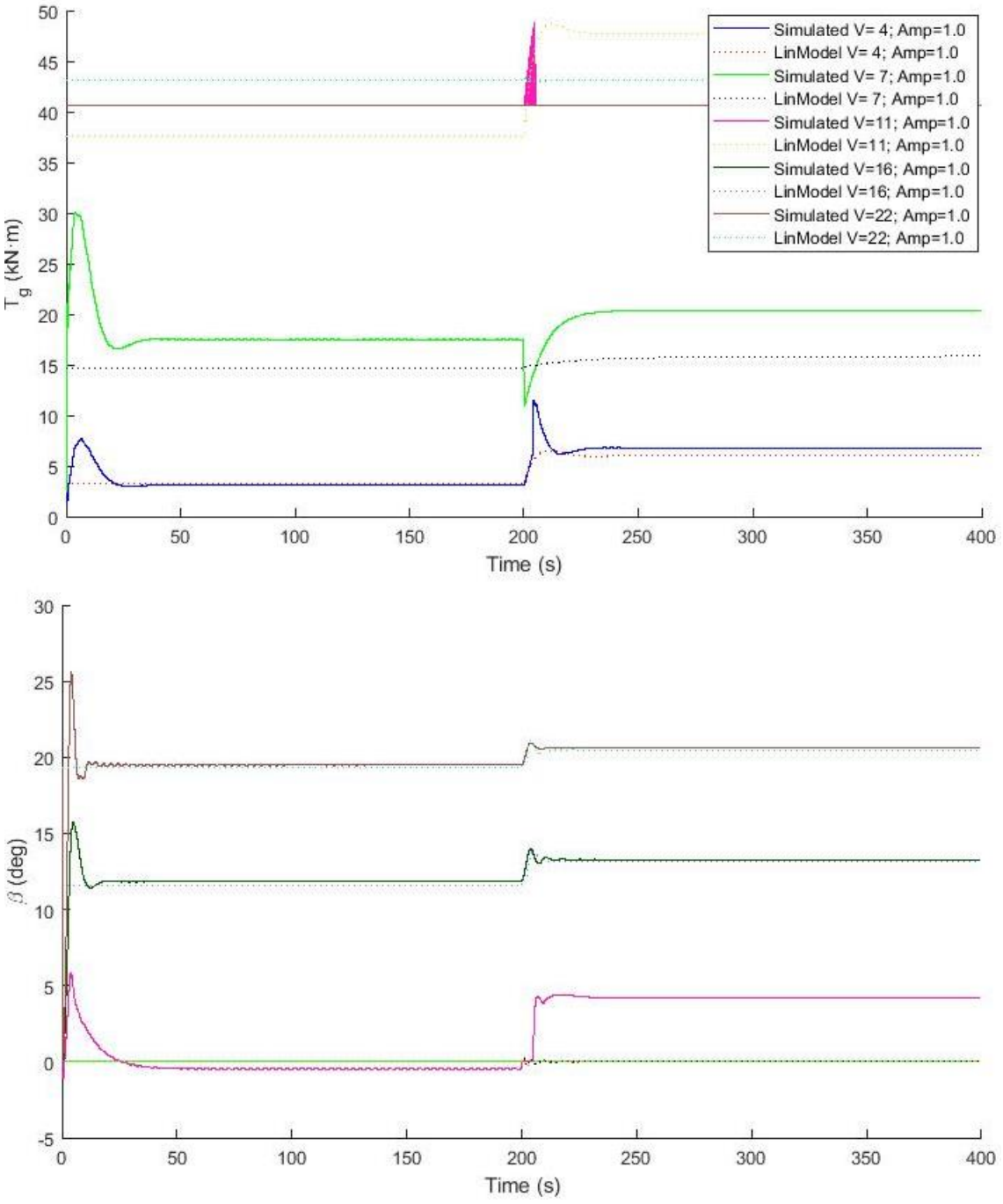

Figura 85. Escalón de viento en lazo cerrado. Tg y β

En la Figura 86 se observa que el modelo lineal a 11 m/s, al haber realizado control de par, ha mantenido la velocidad de giro del generador a su valor nominal mientras que el par ha superado su valor máximo, por lo que se supera la potencia nominal del aerogenerador.

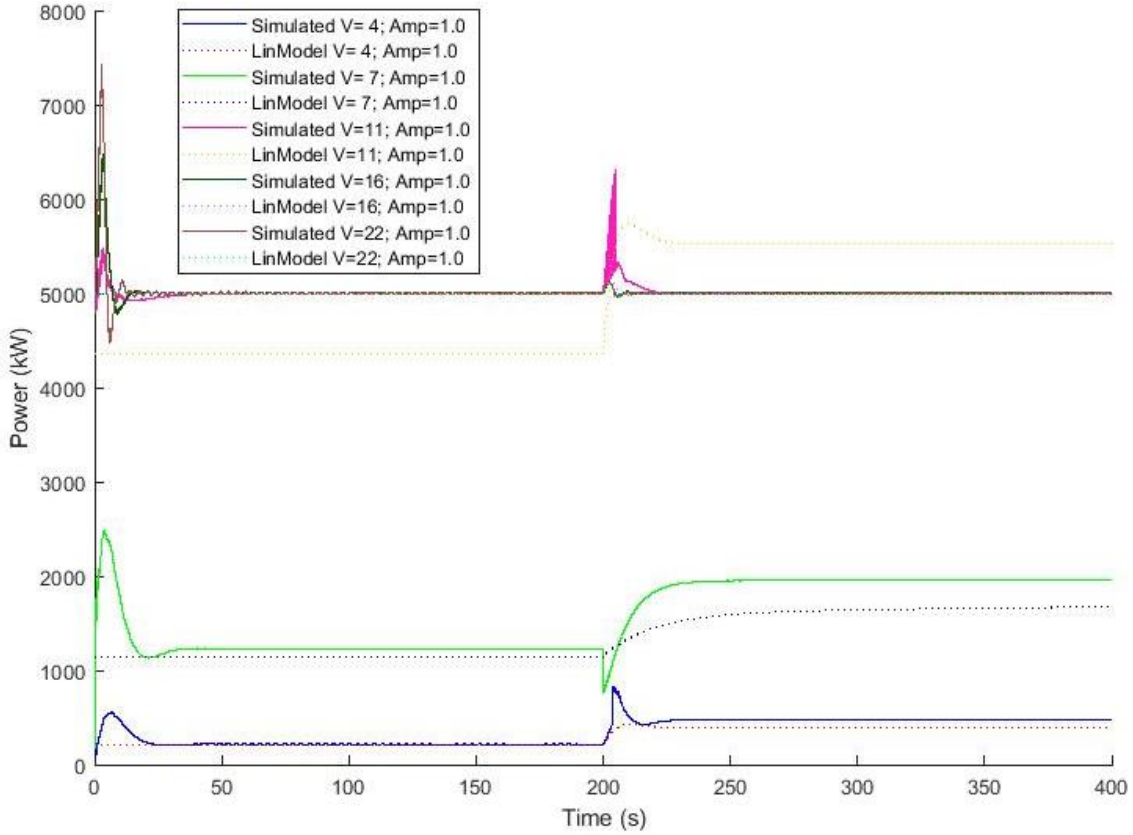

Figura 86. Escalón de viento en lazo cerrado. Potencia

Se introduce un **escalón de pitch** de amplitud 0.02 radianes cuando la máquina está trabajando a 13, 16, 19 y 22 m/s (Figura 87). Tras los instantes iniciales en los que el modelo no lineal oscila hasta encontrar el punto de equilibrio, ambos modelos obtienen resultados muy similares. En ambos casos se amortigua el escalón de pitch que ha sido introducido como una perturbación, no como una señal de control. Este tipo de perturbaciones podrían venir, por ejemplo, ocasionadas por un mal estado del sensor.

En la Figura 88 se observa que el punto de equilibrio de par es diferente entre el modelo lineal y el no lineal. Sin embargo, la dinámica es la misma, ya que al trabajar en zona nominal, el control que actúa es el control de pitch. En dicha figura también se observa la salida del ángulo de pitch que amortigua la perturbación introducida.

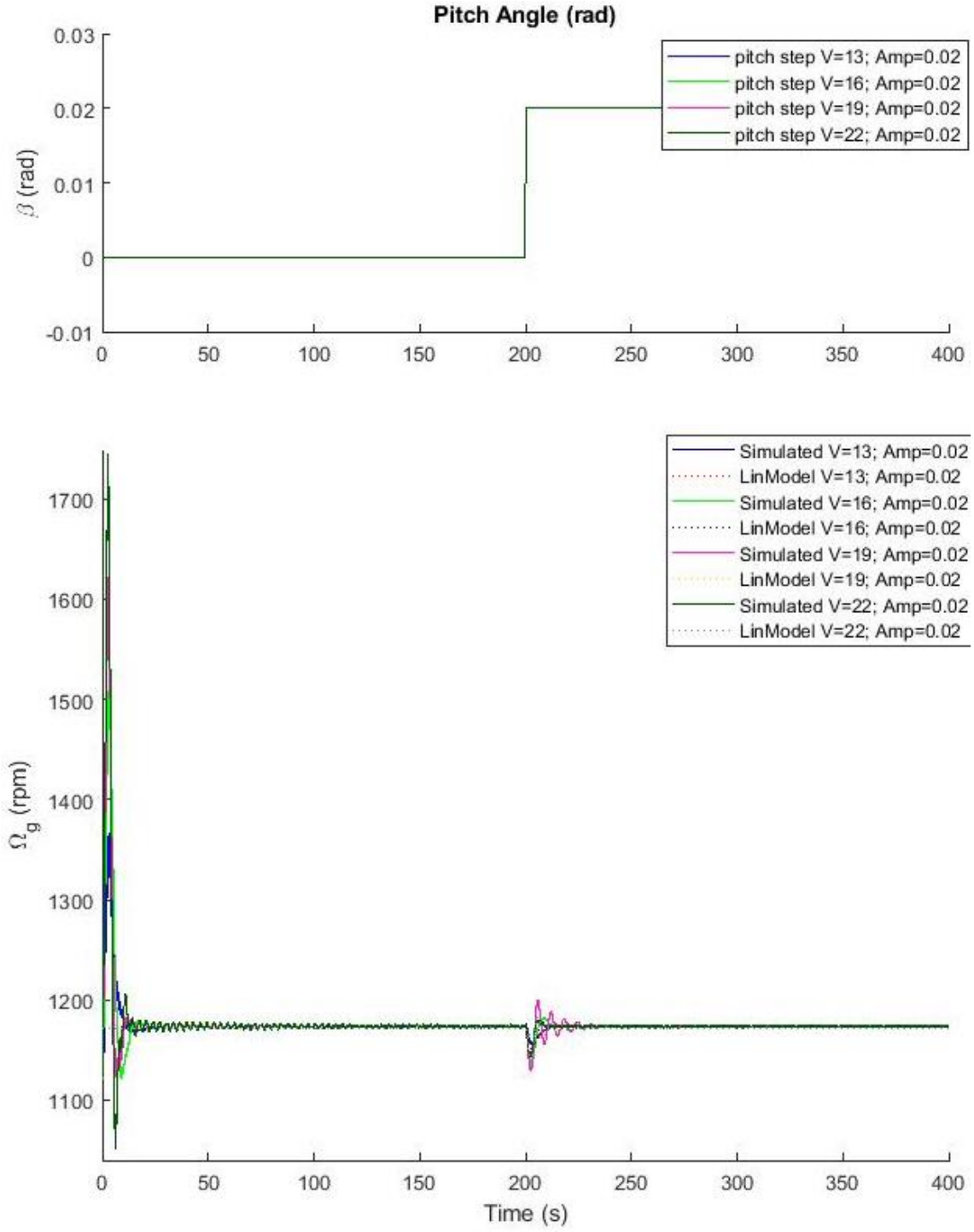

Figura 87. Escalón de pitch en lazo cerrado. Ω_g

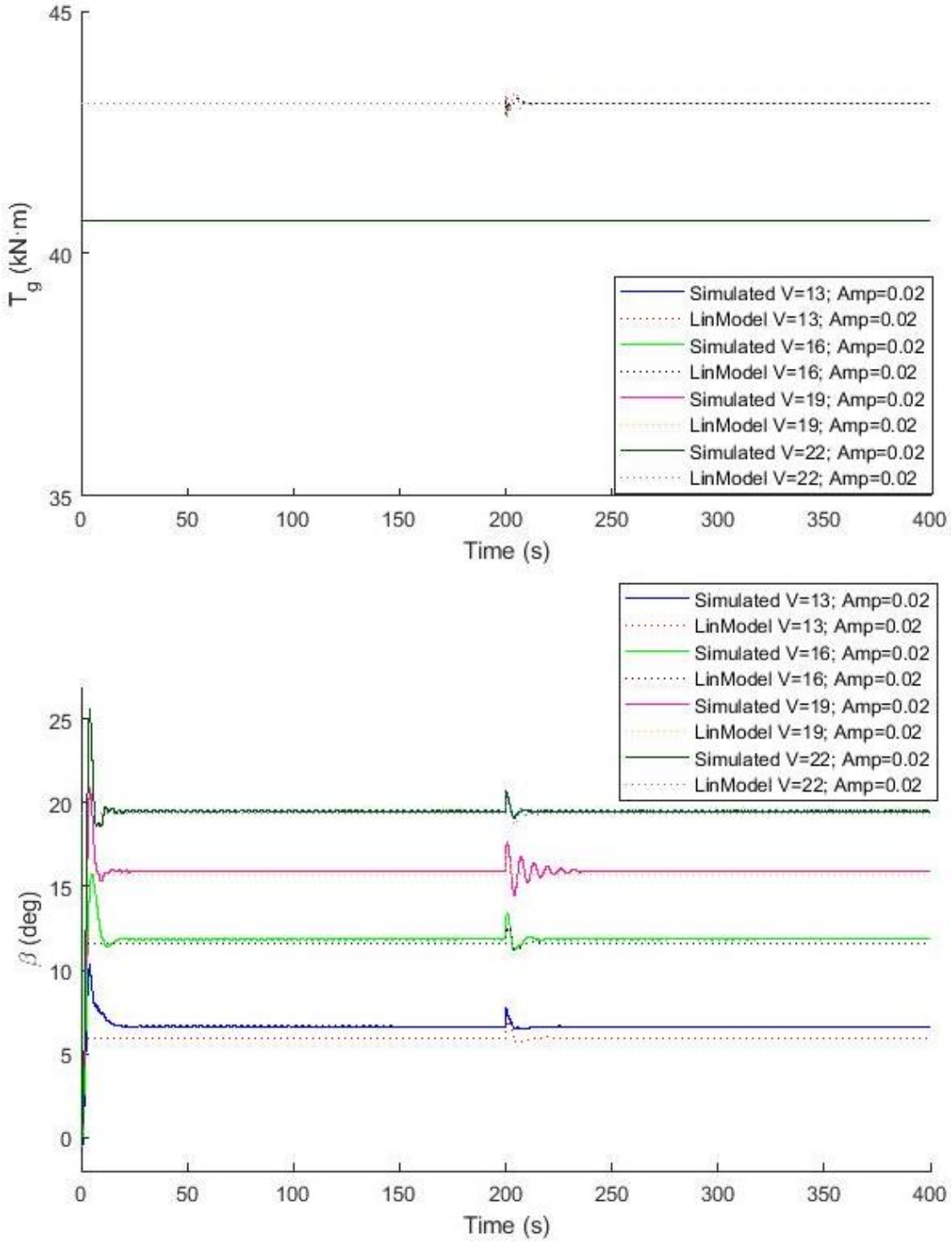

Figura 88. Escalón de pitch en lazo cerrado. Tg y β

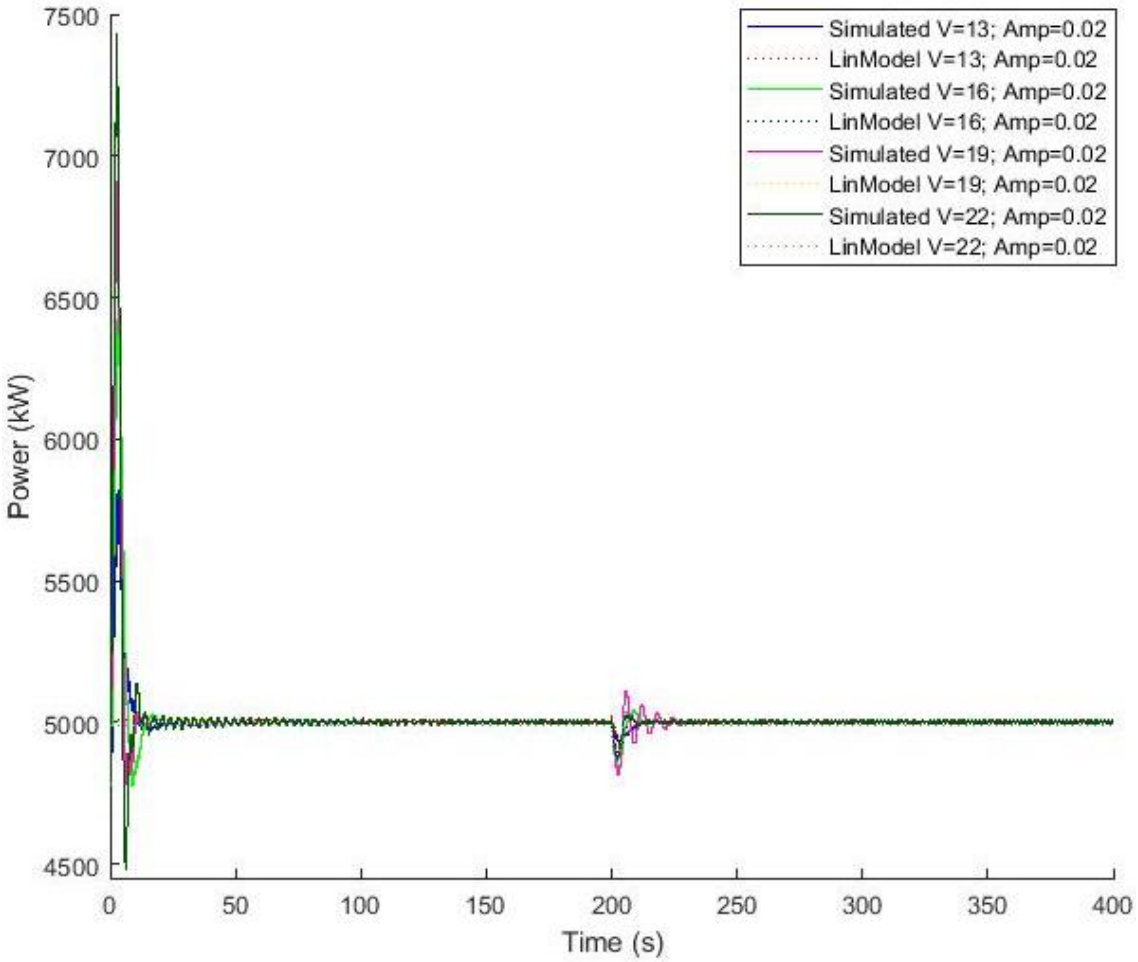

Figura 89. Escalón de pitch en lazo cerrado. Potencia

Se introduce un **escalón de par** de amplitud 1000 kN·m cuando la máquina está trabajando a 4, 6, 8 y 10 m/s (Figura 90). El modelo lineal a 8 y 10 m/s parte de un punto de equilibrio diferente al modelo no lineal, tanto en la velocidad del generador (Figura 90), como el par real del generador (Figura 91) y por lo tanto la potencia resultante (Figura 92).

Al introducir el escalón de par como una perturbación, el modelo no lineal distingue entre comportamiento en la zona 1.5 (4 y 6 m/s) y comportamiento en la zona 2 (8 y 10 m/s). Cuando identifica que la máquina está trabajando en la zona 2 o zona cuadrática, el nuevo punto de equilibrio tras el escalón se corresponde con una velocidad de giro del generador entre el valor mínimo y el nominal. Sin embargo, el valor de la velocidad de giro del generador debería haber sido esa desde el inicio de la simulación, ya que aunque no tiene porqué ser exactamente igual a la del modelo lineal, sí que debería ser similar. Este error puede deberse a que el valor inicial dado en la simulación se correspondía con una velocidad de giro mínima cuando debería haber sido la velocidad de giro del generador del punto de equilibrio del modelo lineal a esa velocidad de viento.

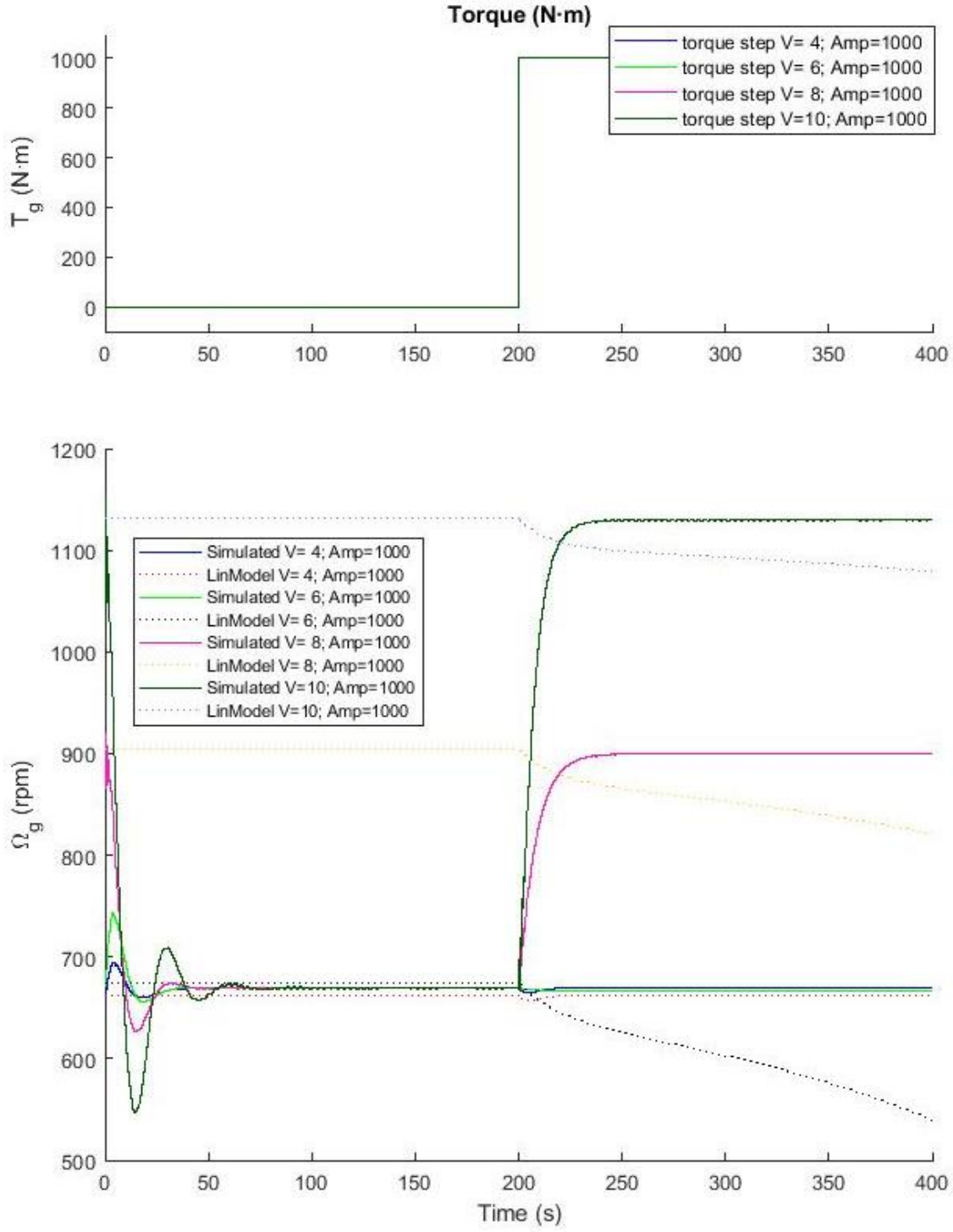

Figura 90. Escalón de par en lazo cerrado. Ω_g

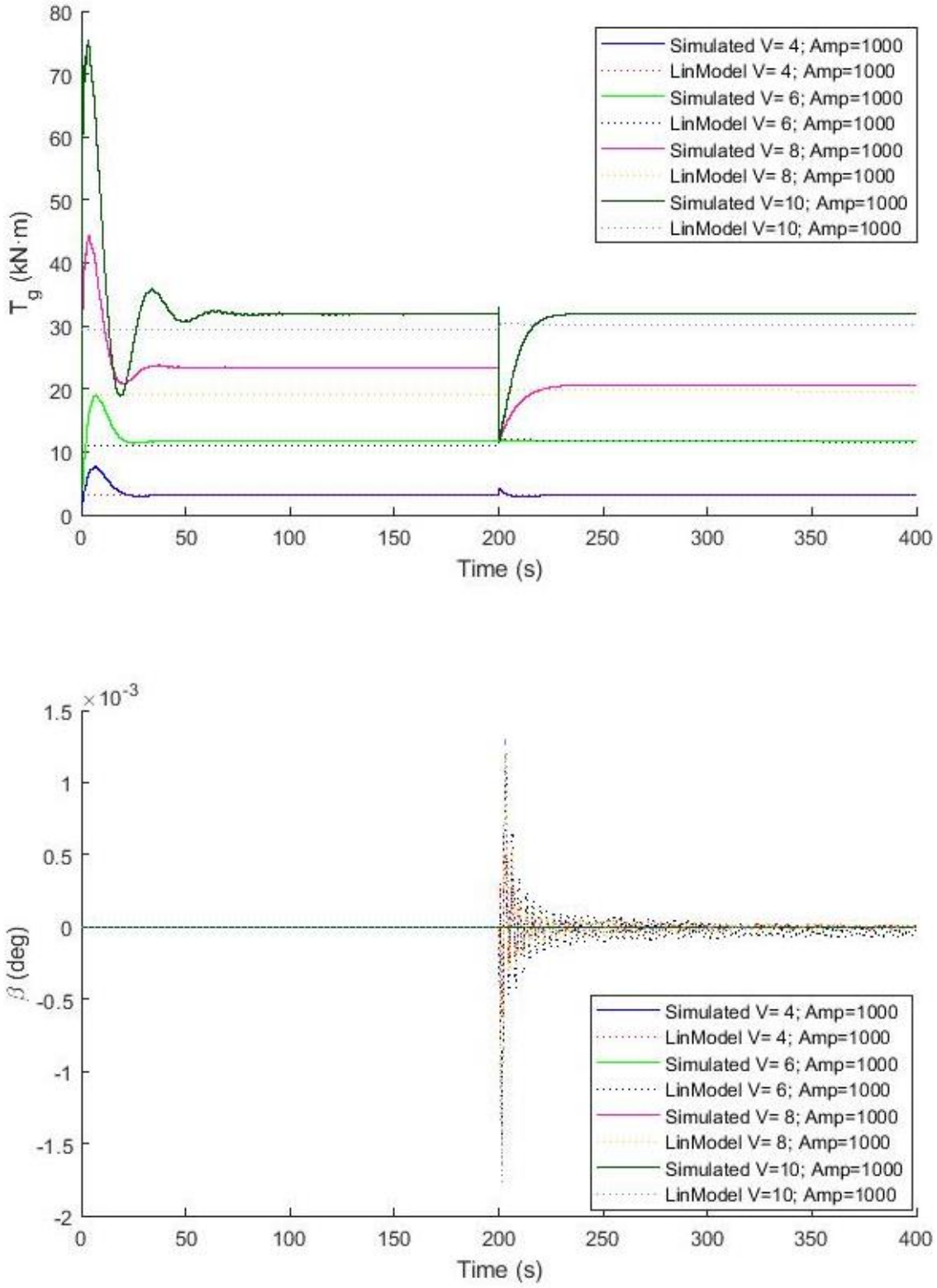

Figura 91. Escalón de par en lazo cerrado. Tg y β

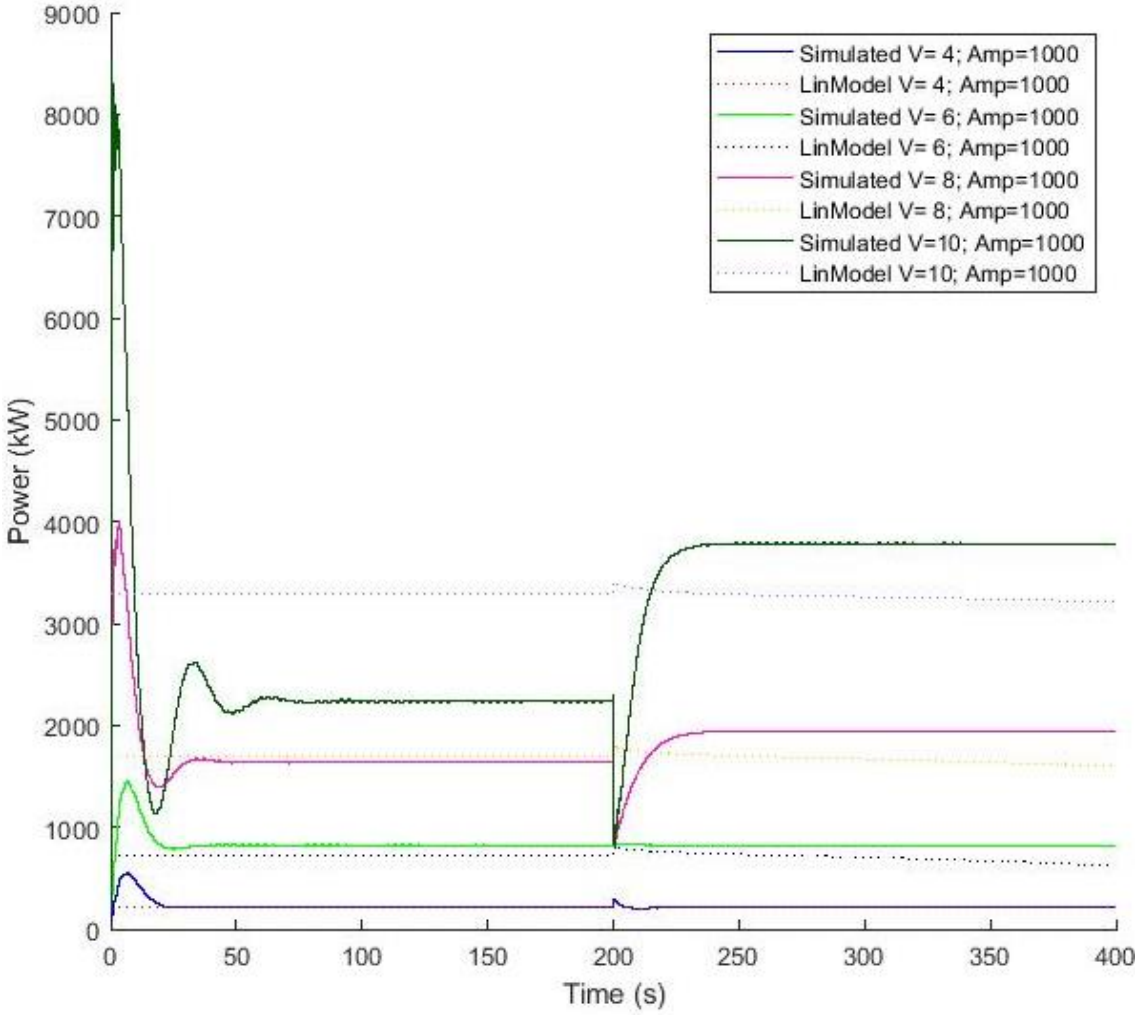

Figura 92. Escalón de par en lazo cerrado. Potencia

Rampa

Se introduce una **rampa de viento** de amplitud 3 m/s cuando la máquina está trabajando a 4, 7, 11, 16 y 22 m/s (Figura 93). Cuando la rampa de viento empieza en zona nominal, el comportamiento del modelo lineal es muy similar al del modelo no lineal, tanto en mantener la velocidad a su valor nominal (Figura 93), como en el valor de pitch "real" (Figura 94) y la potencia obtenida (Figura 95).

Sin embargo, cuando el sistema empieza en zona de control de par, el modelo lineal difiere del modelo no lineal. En el caso de 4 m/s, el modelo lineal se comporta como zona 1.5 a lo largo de toda la simulación, mientras que el modelo no lineal, pasada la mitad de la simulación, identifica una transición a la zona cuadrática por lo que la máquina se acelera. Este comportamiento del modelo lineal es esperable ya que se emplea el modelo de la turbina de 4 m/s para toda la simulación.

En cuanto a la rampa de 7 m/s, el modelo lineal se acelera más rápidamente que el modelo no lineal debido a que el par impuesto tiene menor pendiente. Esto se debe a que el modelo de la turbina empleado se corresponde con el de 7 m/s mientras que al final de la simulación debería ser el de 10 m/s. En dicho rango de viento, el modelo de la turbina varía mucho, por lo que es esperable que el comportamiento no se ajuste al del modelo no lineal.

A 11 m/s, el modelo lineal trabaja en la segunda vertical con lo que hace control de par superando incluso el valor de par nominal pero manteniendo la velocidad de giro del generador a su valor nominal. Esto se traduce en una potencia superior a la nominal. En cuanto al modelo lineal, tras la transición inicial, el valor del viento ya se corresponde con zona nominal, por lo que hace control de pitch, manteniendo el par a su valor nominal y por lo tanto también su potencia.

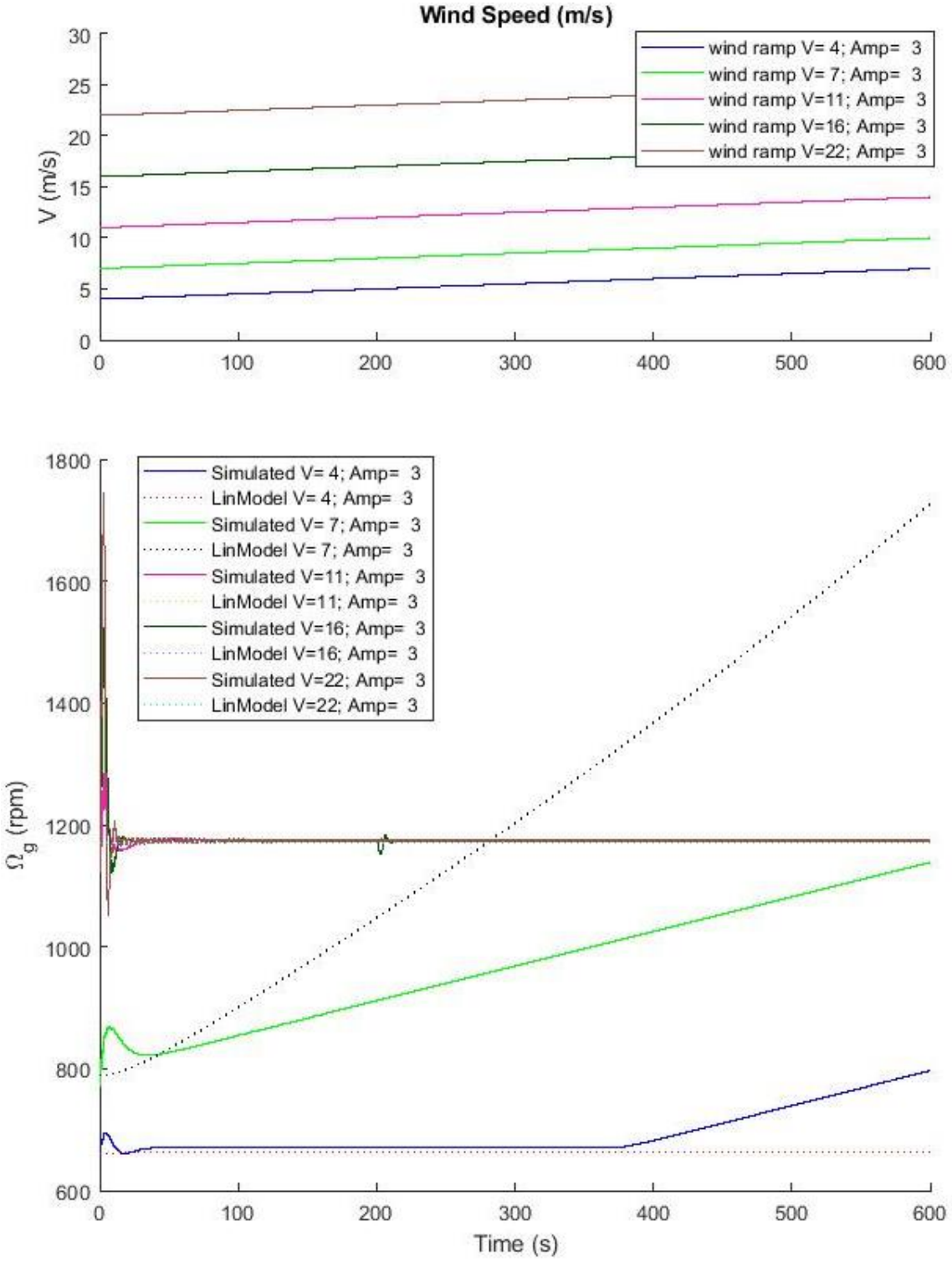

Figura 93. Rampa de viento en lazo cerrado. Ω_g

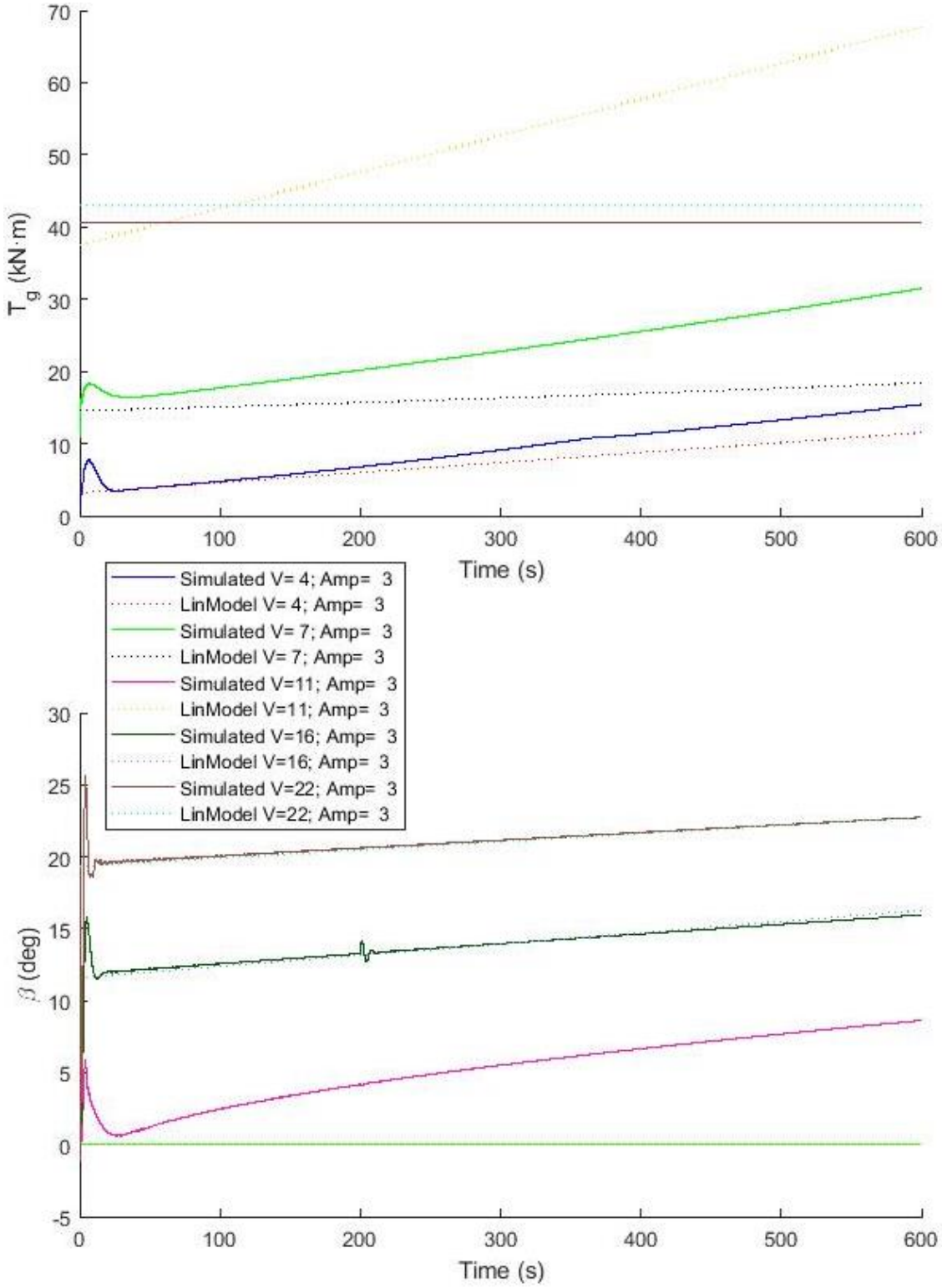

Figura 94. Rampa de viento en lazo cerrado. Tg y β

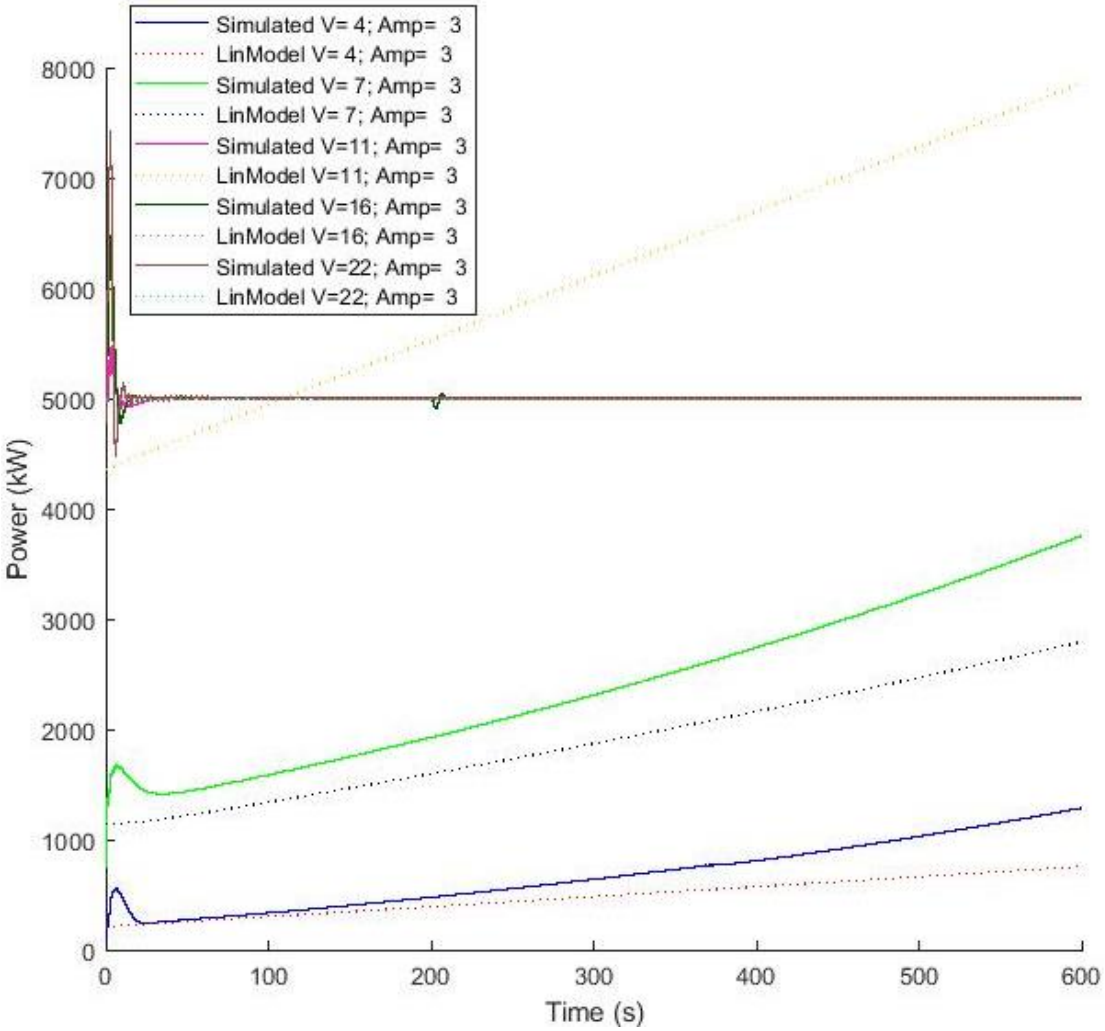

Figura 95. Rampa de viento en lazo cerrado. Potencia

Se introduce una **rampa de pitch** de amplitud 0.1 radiantes cuando la máquina está trabajando a 13, 16, 19 y 22 m/s (Figura 96). Tanto el modelo lineal como el no lineal tienen un comportamiento similar: mantienen la velocidad de giro del generador a su valor nominal (Figura 96), el par se fija constante a su valor nominal al estar trabajando en zona nominal con control de pitch (Figura 97) y la potencia se mantiene constante a 5 MW (Figura 98).

Hay un pequeño desvío en los valores del ángulo de pitch entre el modelo lineal y el no lineal, pero es una desviación mínima y la dinámica es similar (Figura 97).

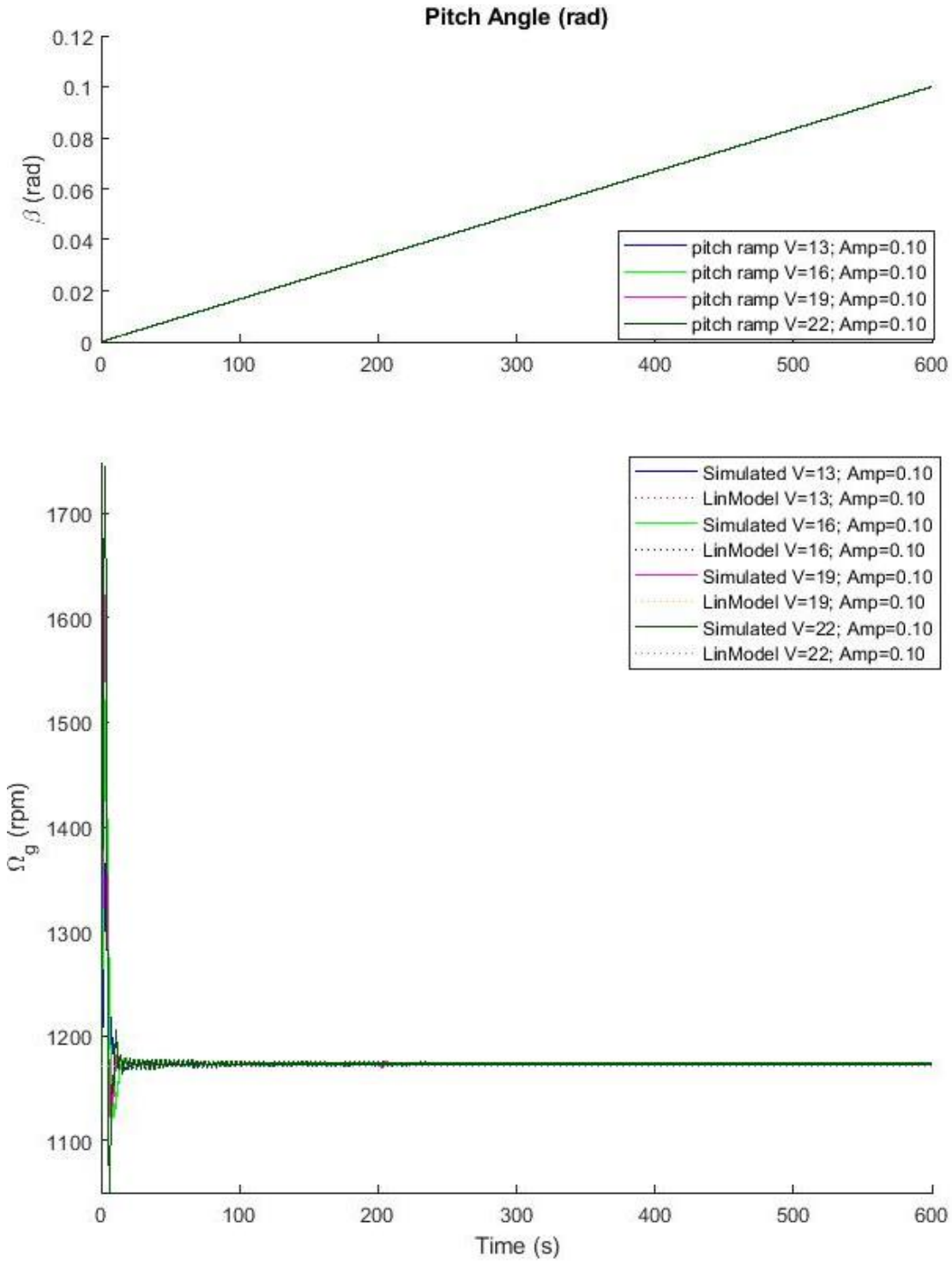

Figura 96. Rampa de pitch en lazo cerrado. Ω_g

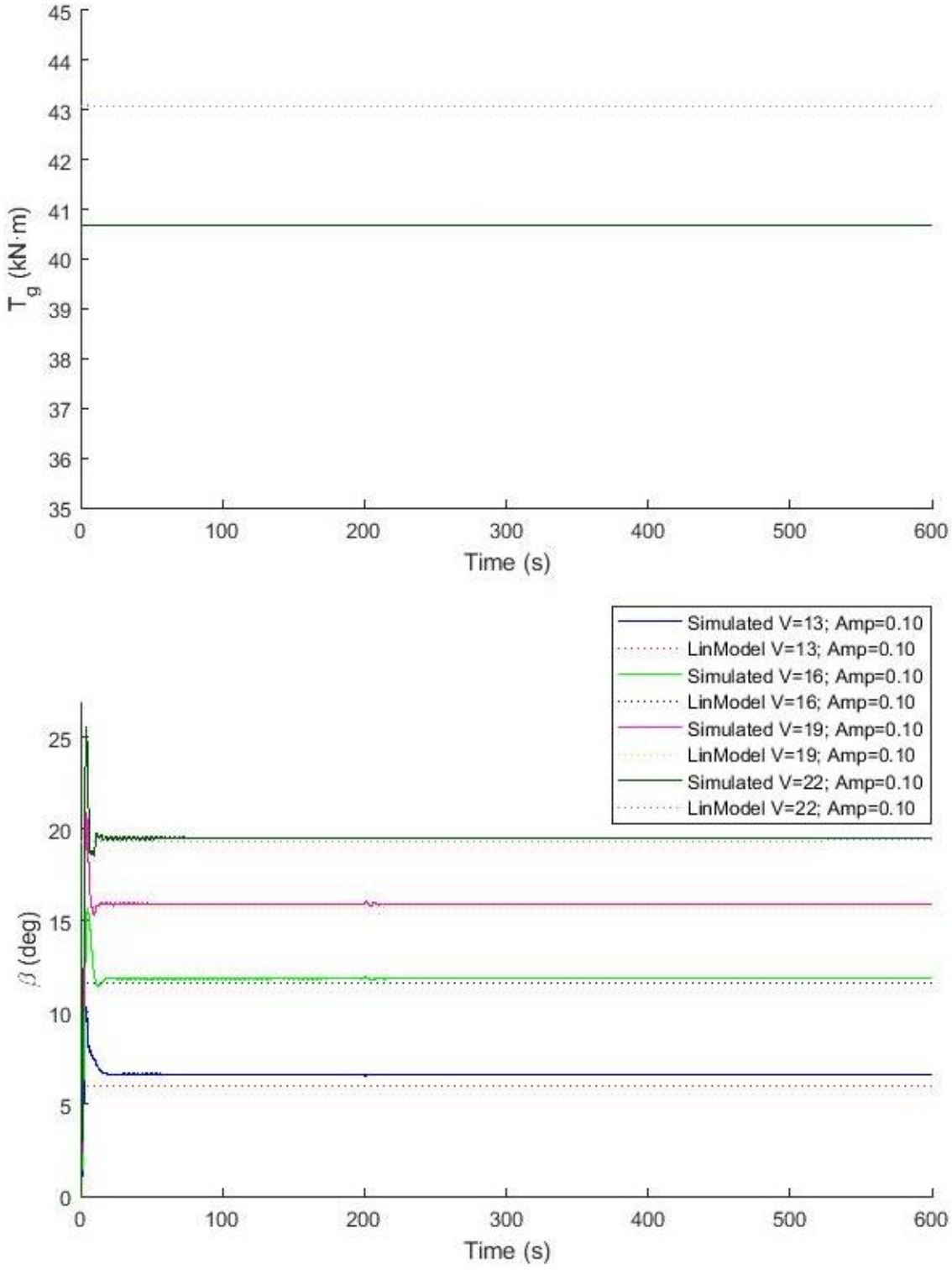

Figura 97. Rampa de pitch en lazo cerrado. Tg y β

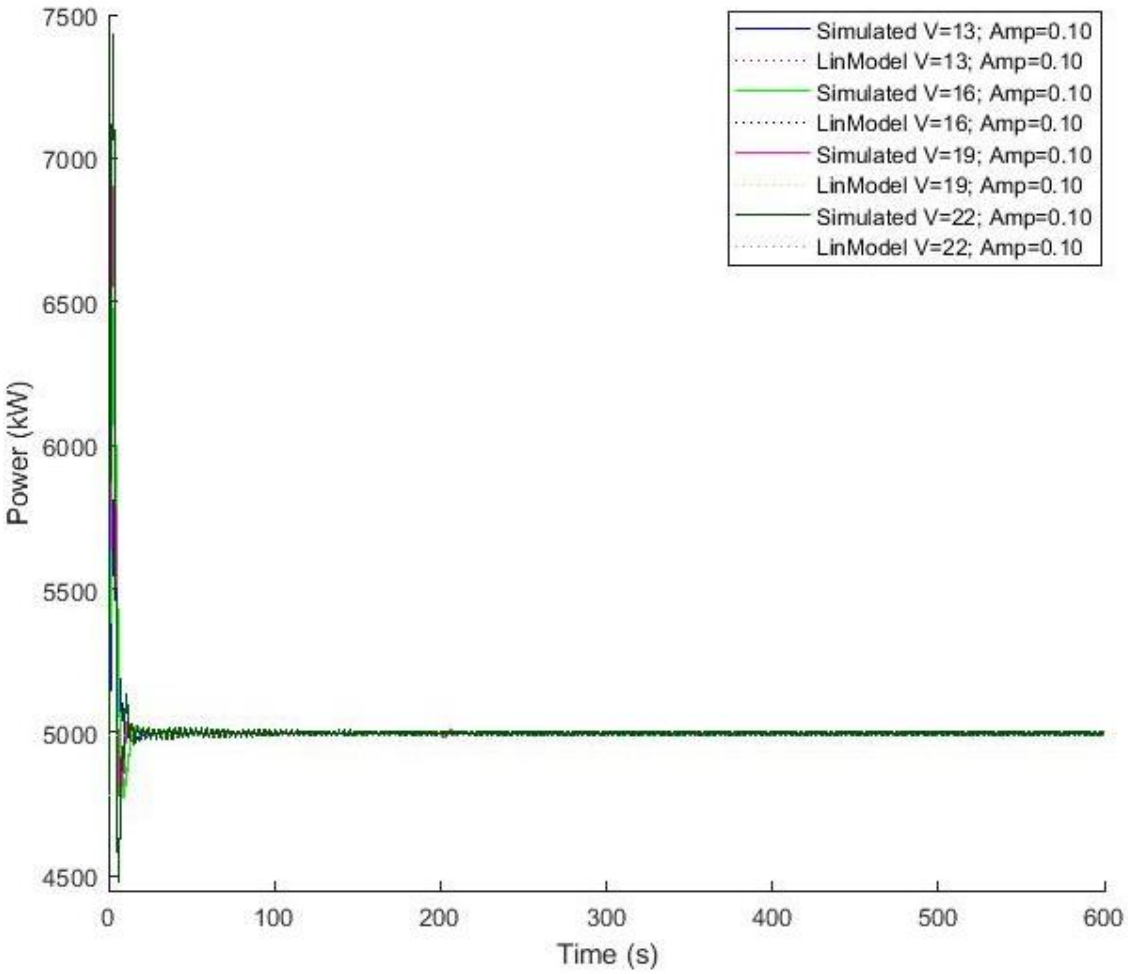

Figura 98. Rampa de pitch en lazo cerrado. Potencia

Se introduce una perturbación del tipo **rampa de par** de amplitud 2000 N·m cuando la máquina está trabajando a 4, 6, 8 y 10 m/s (Figura 99). Al trabajar en la zona de control de par, el valor del ángulo de pitch es nulo (Figura 100).

La máquina en ambos modelos se frena, pero en el caso del modelo lineal lo hace más rápido y parece inestable (Figura 99). Esto se debe a que el par del generador en el caso del modelo no lineal se opone a la perturbación de par, mientras que en el modelo lineal, comienza oponiéndose al par perturbación pero finalmente el par adquiere pendiente negativa, especialmente a 6 m/s (Figura 100).

A 4 m/s el modelo lineal y el modelo no lineal se asemejan mucho, tanto en dinámica como en punto de equilibrio. Esto se debe a que en todo momento la máquina está trabajando en la primera vertical y no hay cambio de zona. Sin embargo, a 6 m/s la máquina se encuentra en estado de transición entre la primera vertical y la zona cuadrática, por lo que es normal que el comportamiento no sea el ideal, sino que aparezcan problemas en la turbina.

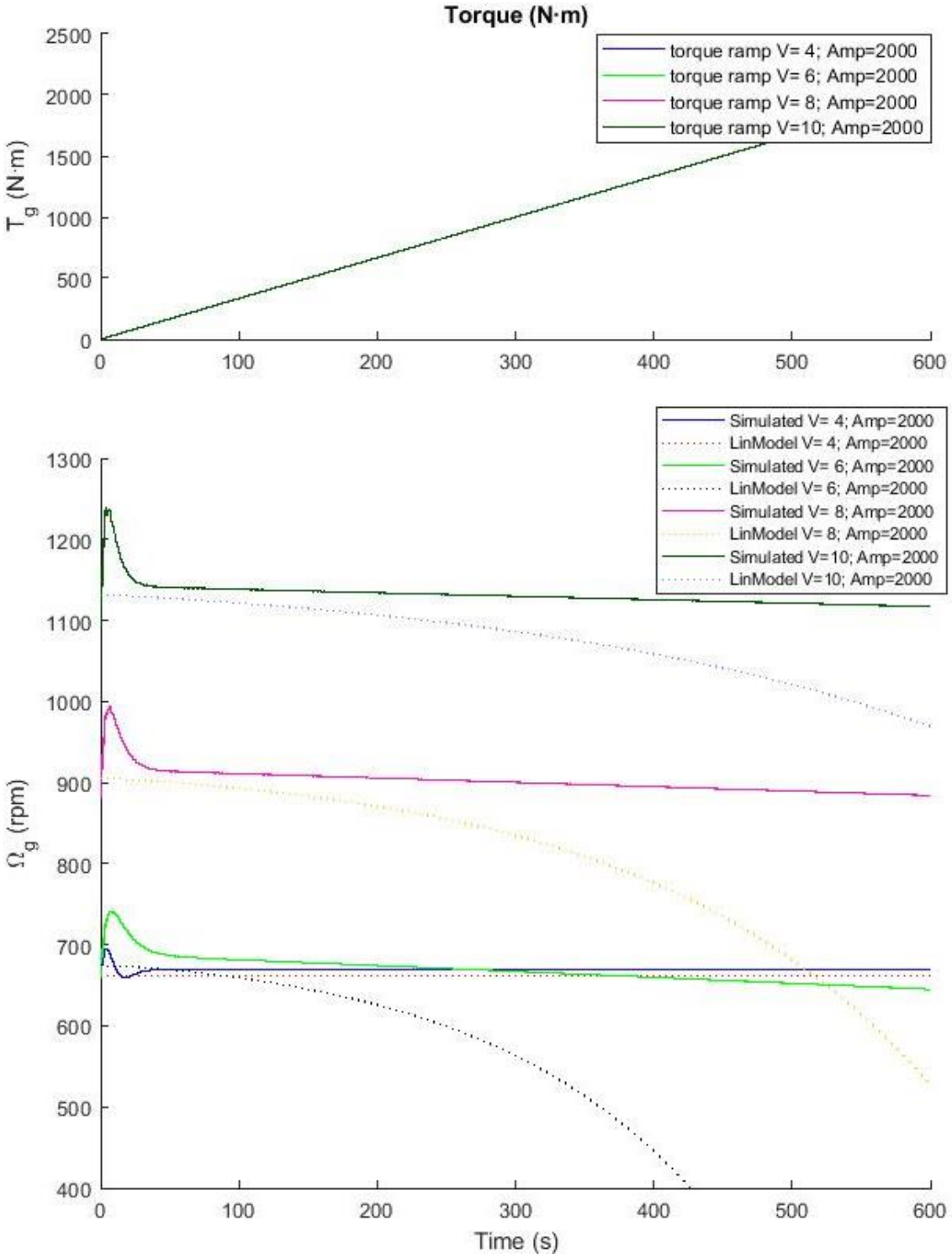

Figura 99. Rampa de par en lazo cerrado. Ω_g

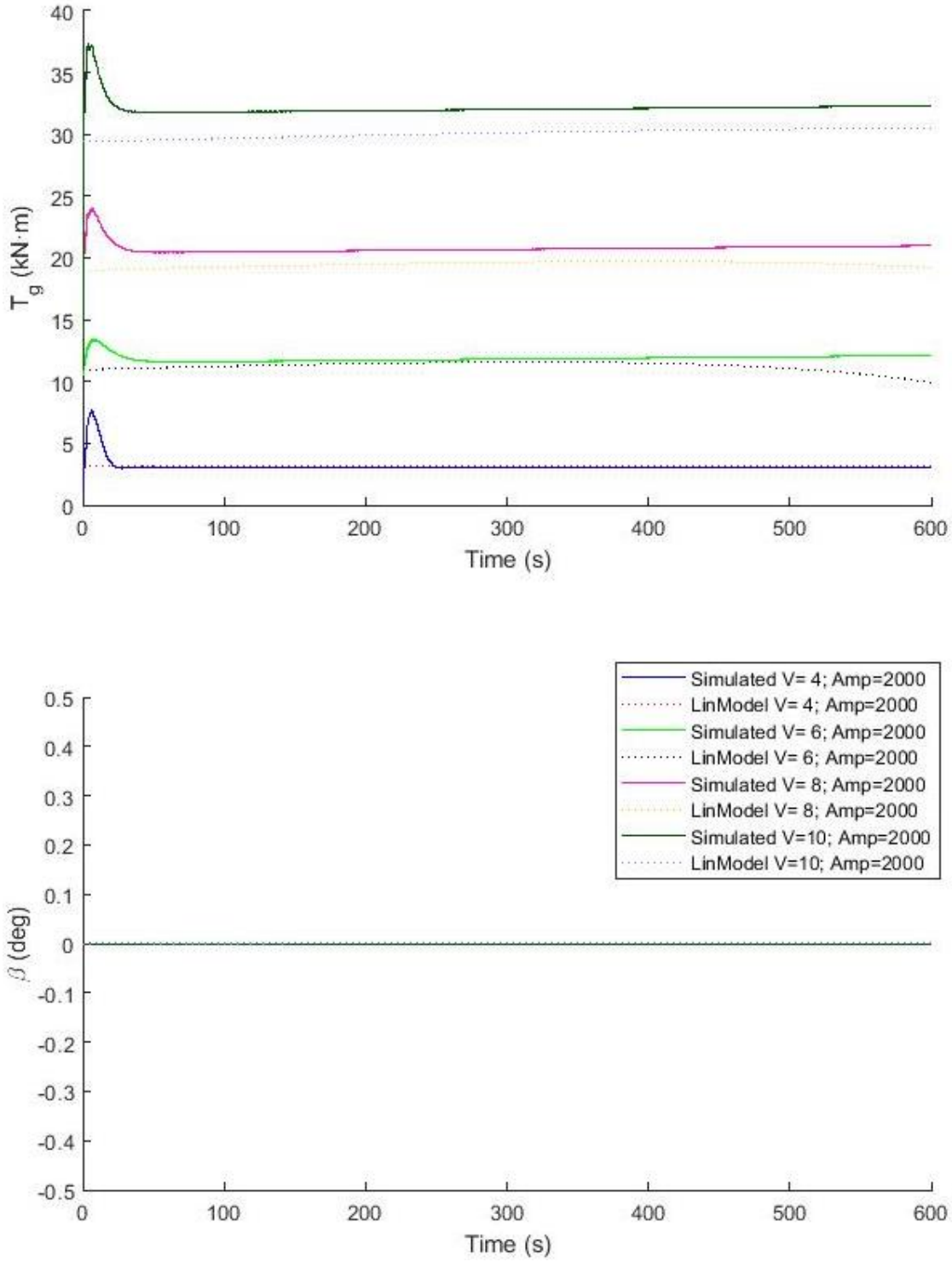

Figura 100. Rampa de par en lazo cerrado. Tg y β

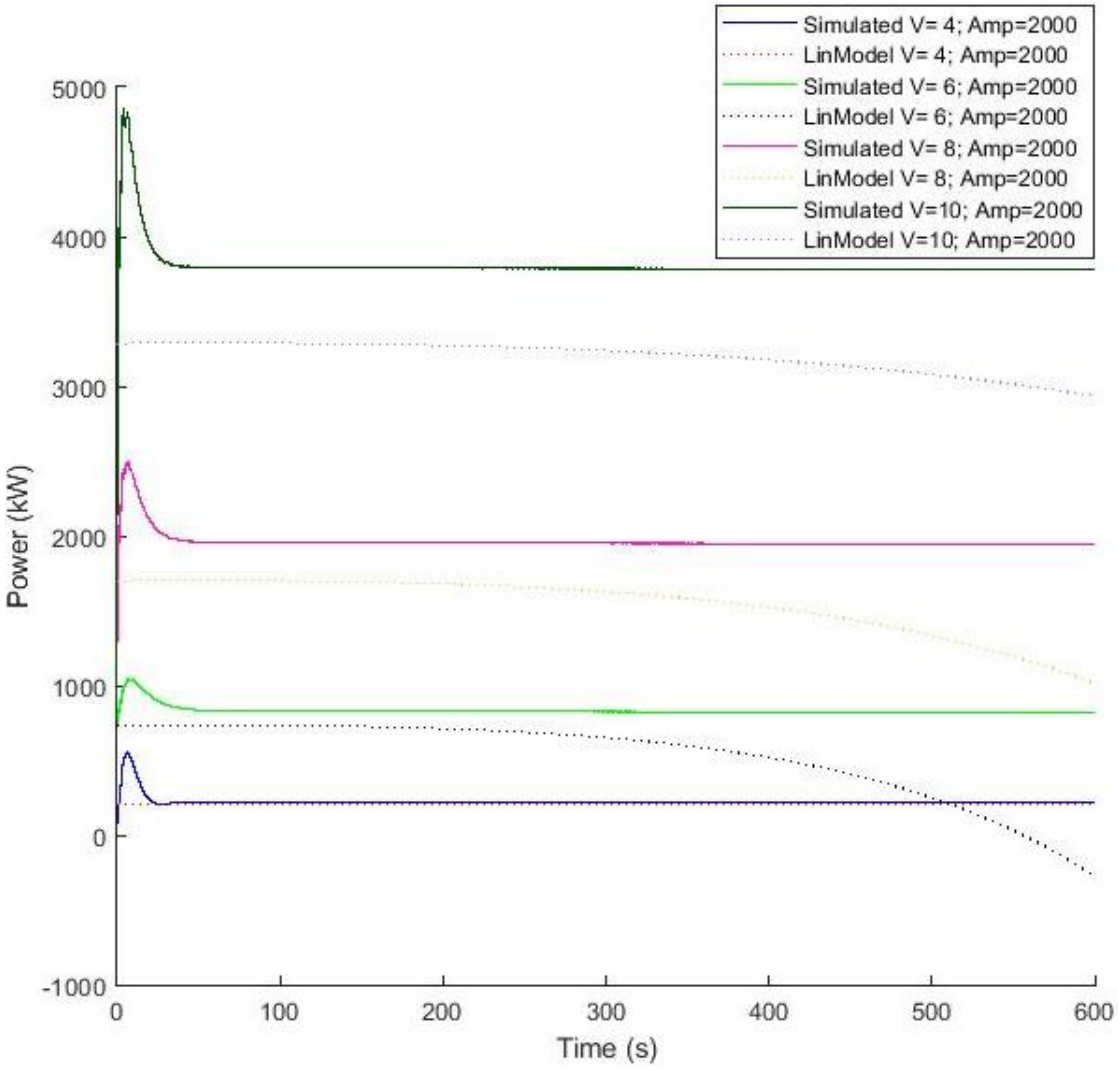

Figura 101. Rampa de par en lazo cerrado. Potencia

Sinusoide

Al introducir una señal de viento senoidal centrada en 11 m/s y de amplitud 1 m/s, la máquina no se comporta como se esperaba. Esto se debe a que el sistema está continuamente cambiando de zonas de operación, el sistema se desestabiliza y no se obtiene la señal senoidal esperada de salida.

Este problema de transición entre zonas encuentra su punto crítico en 11 m/s. Por ello, al introducir una señal senoidal de 1 m/s de amplitud y centrada en 9 m/s se obtiene un buen resultado, mientras que a 10 m/s se perturba la señal de salida.

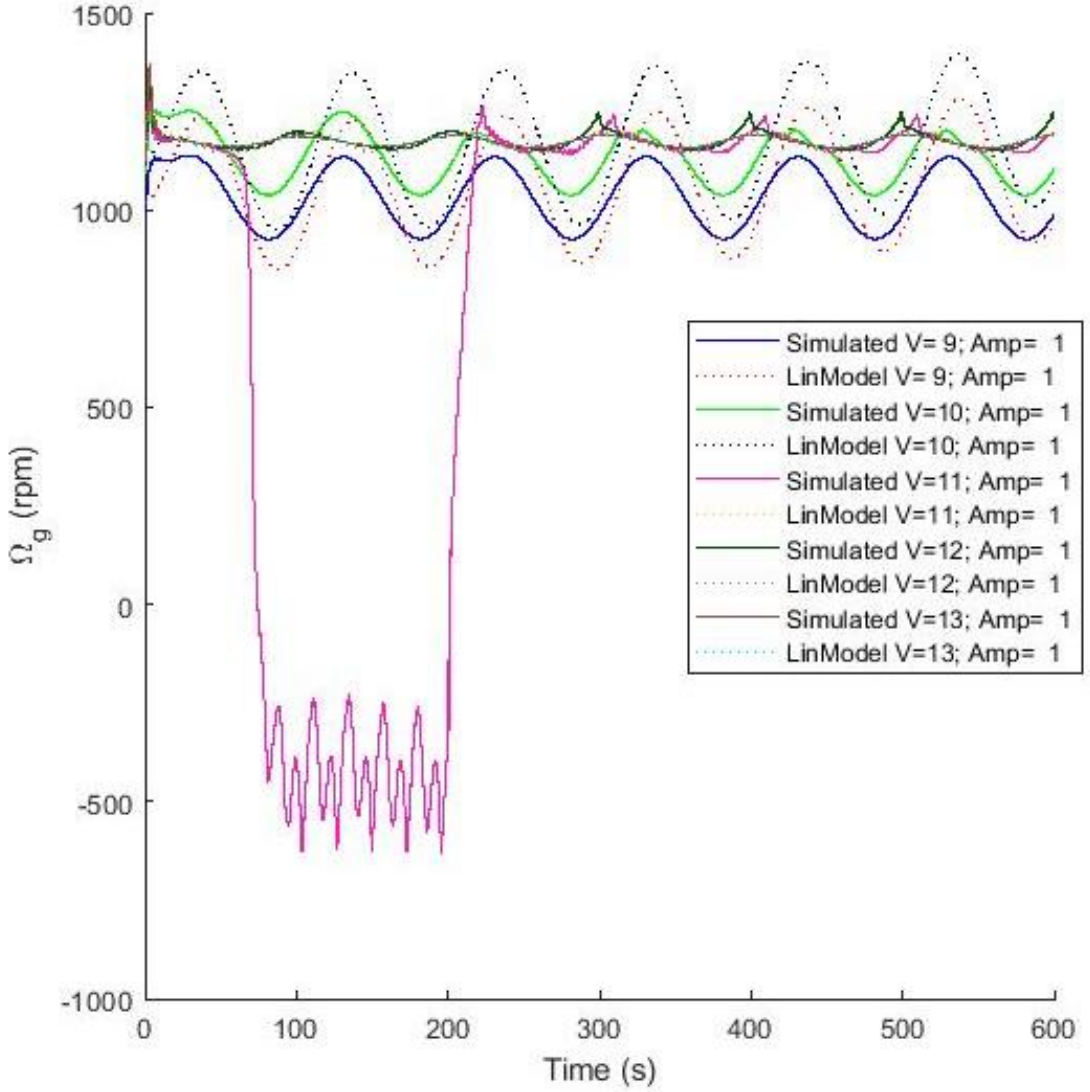

Figura 102. Seno de viento a 9, 10, 11 y 12 m/s.

La transición entre la segunda vertical y la zona nominal es comúnmente fuente de problemas en el diseño y operación de los aerogeneradores. En las anteriores figuras de rampas y escalones en torno a dicha velocidad de viento no se han encontrado grandes problemas. Sin embargo, al introducir una señal senoidal se está conmutando continuamente entre zonas y tipos de control, se acentúa el efecto de la velocidad de viento crítica.

A nivel comercial, cada empresa sigue unas normas y un criterio para dicha transición. Sin embargo, dicho protocolo es específico de cada empresa y forma parte de su 'saber hacer', por lo que suele conllevar una política de confidencialidad asociada.

Ya que el propósito de este trabajo no es mejorar específicamente dicha transición ante señales oscilantes a la velocidad crítica del aerogenerador, no se le va a dar mayor importancia. Sin embargo, como trabajo futuro podría mejorarse la transición y crear un protocolo ante dicho tipo de situaciones.

Se introduce un **seno de viento** de amplitud 1 m/s cuando la máquina está trabajando a 4, 7, 9, 12, 16 y 22 m/s (Figura 103). A las velocidades de viento correspondientes a control de par (4, 7 y 9 m/s), el comportamiento del modelo lineal y el no lineal es similar, fijando en ambos casos el pitch a 0. La amplitud del par es mayor en el caso del modelo no lineal (Figura 104) lo que

provoca una menor desviación de la velocidad de giro del generador aunque una mayor amplitud en la potencia (Figura 105).

En cuanto a 12 m/s, el modelo lineal se comporta como zona nominal a lo largo de toda la simulación incluso cuando ello implica valores negativos de pitch, mientras que el modelo no lineal varía entre zona 3 y zona 2.5. Esto se debe a que la velocidad de viento que se está simulando se encuentra cercana a la velocidad de viento crítica de transición.

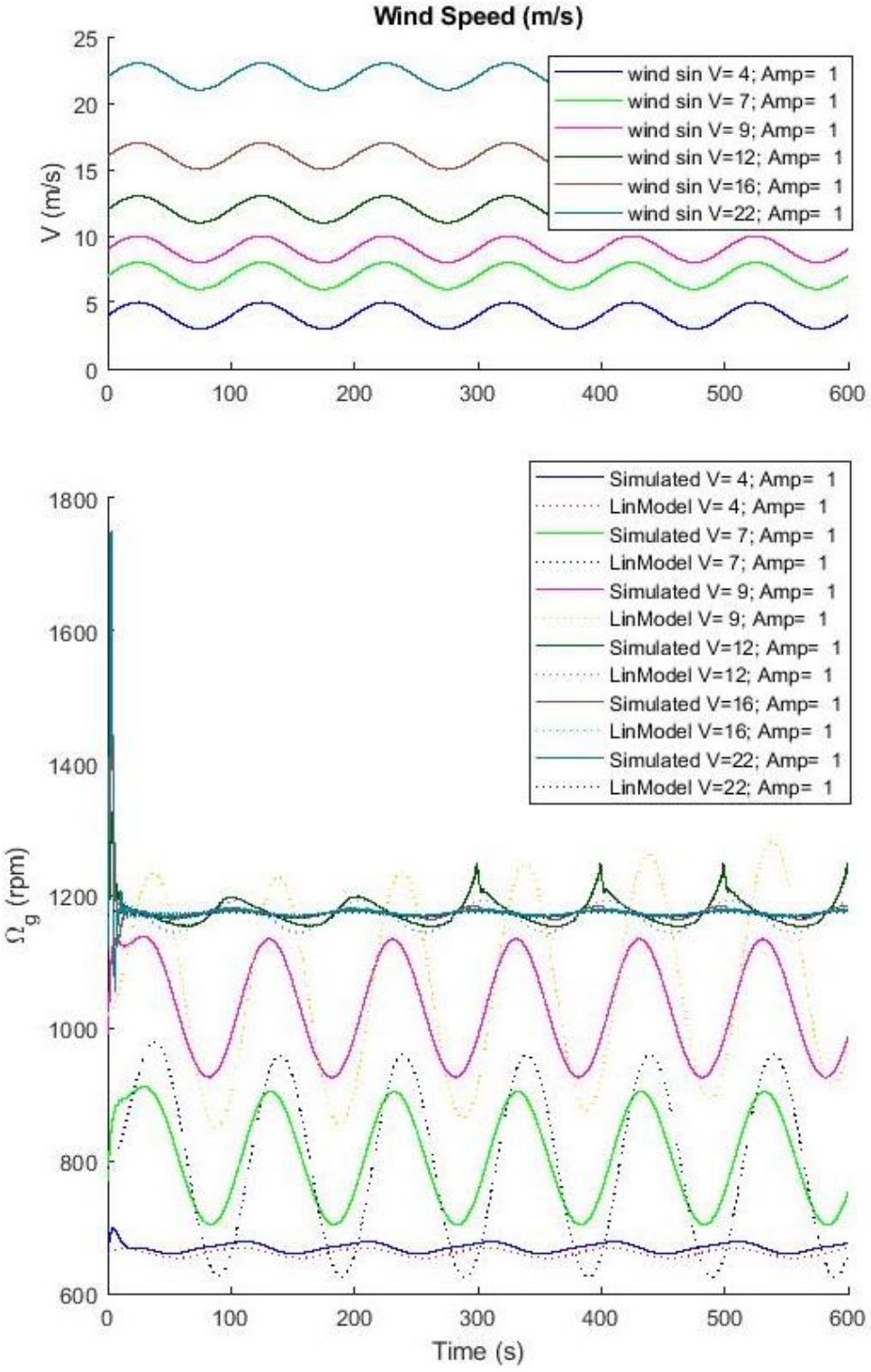

Figura 103.Seno de viento en lazo cerrado. Ω_g

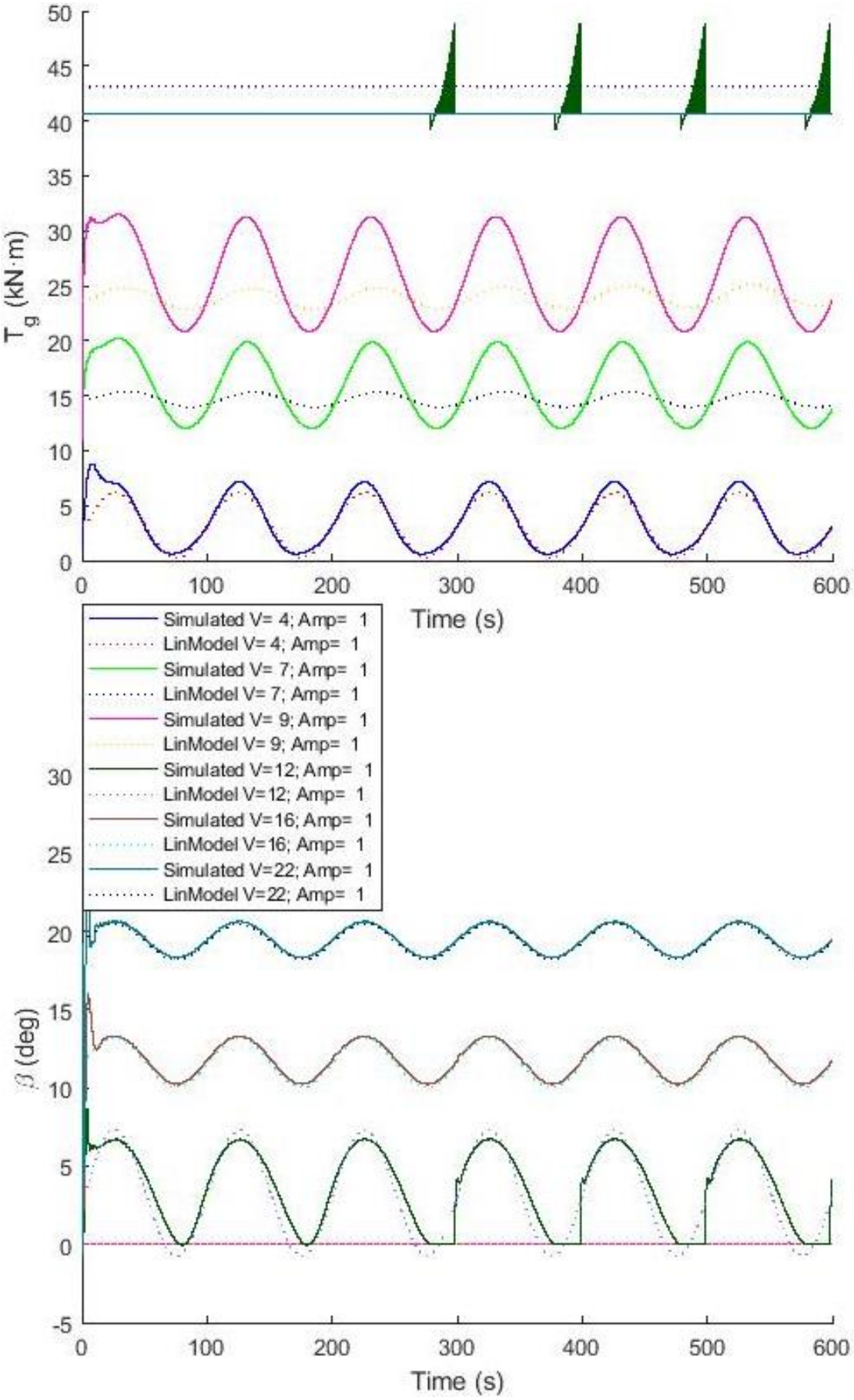

Figura 104. Seno de viento en lazo cerrado. Tg y β

A 16 y 22 m/s el modelo lineal y el modelo no lineal se comportan de manera semejante, tanto dinámicamente como en el punto de equilibrio.

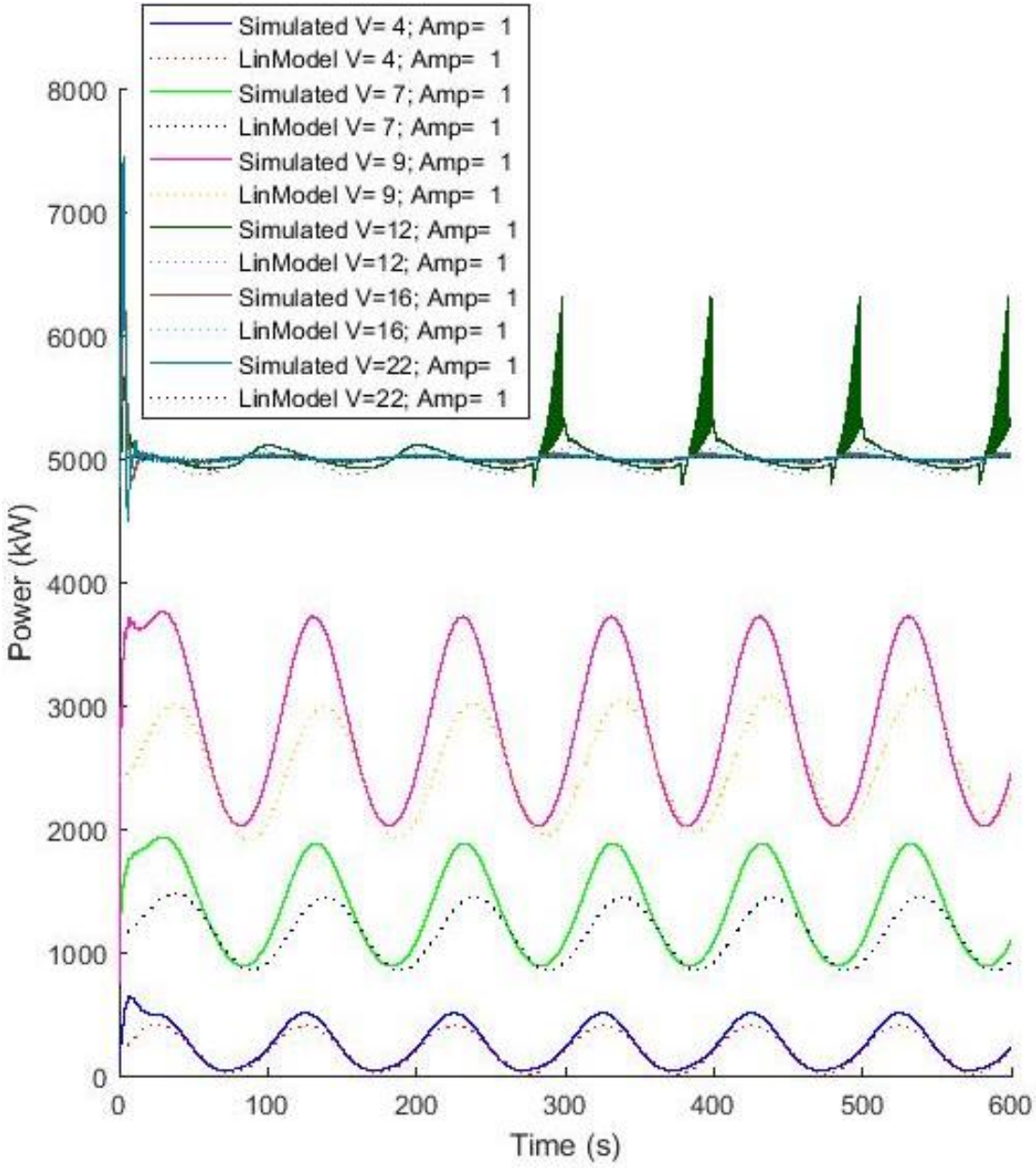

Figura 105. Seno de viento en lazo cerrado. Potencia

Se introduce una perturbación **senoidal de pitch** de amplitud 0.02 radiantes cuando la máquina está trabajando a 13, 16, 19 y 22 m/s (Figura 106). Al corresponderse con velocidades de viento de control de pitch, el par se establece a su valor nominal (Figura 107).

El ángulo de pitch de salida del modelo lineal es similar al del modelo no lineal, tanto dinámicamente como en el punto de equilibrio. La mayor diferencia entre ambos modelos se encuentra en una pequeña variación del punto de equilibrio entre ambos modelos a 13 m/s (Figura 107). Al tener un comportamiento similar de pitch y de velocidad de giro del generador, la potencia obtenida con el modelo lineal es semejante a la del modelo no lineal (Figura 108).

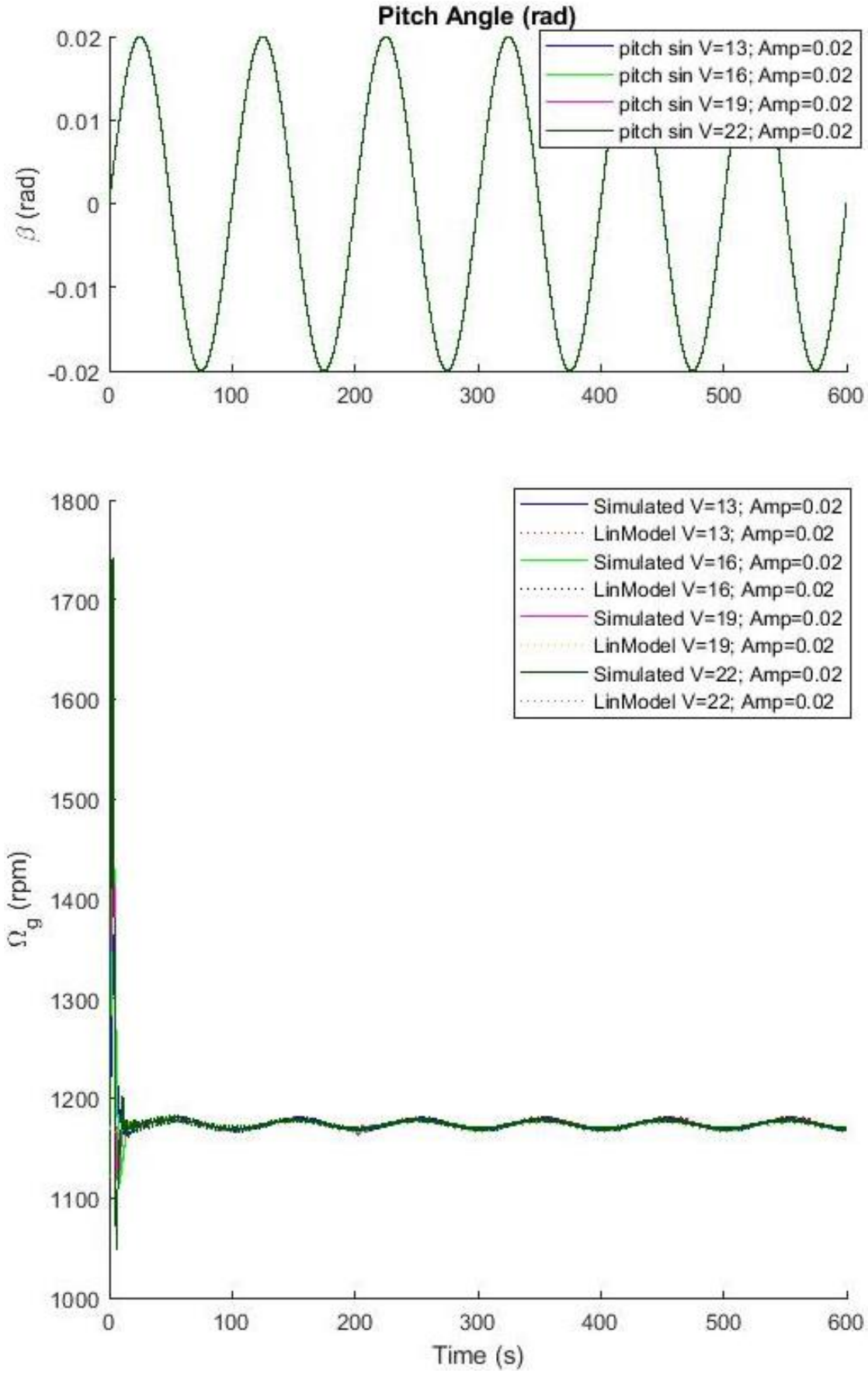

Figura 106. Seno de pitch en lazo cerrado. Ω_g

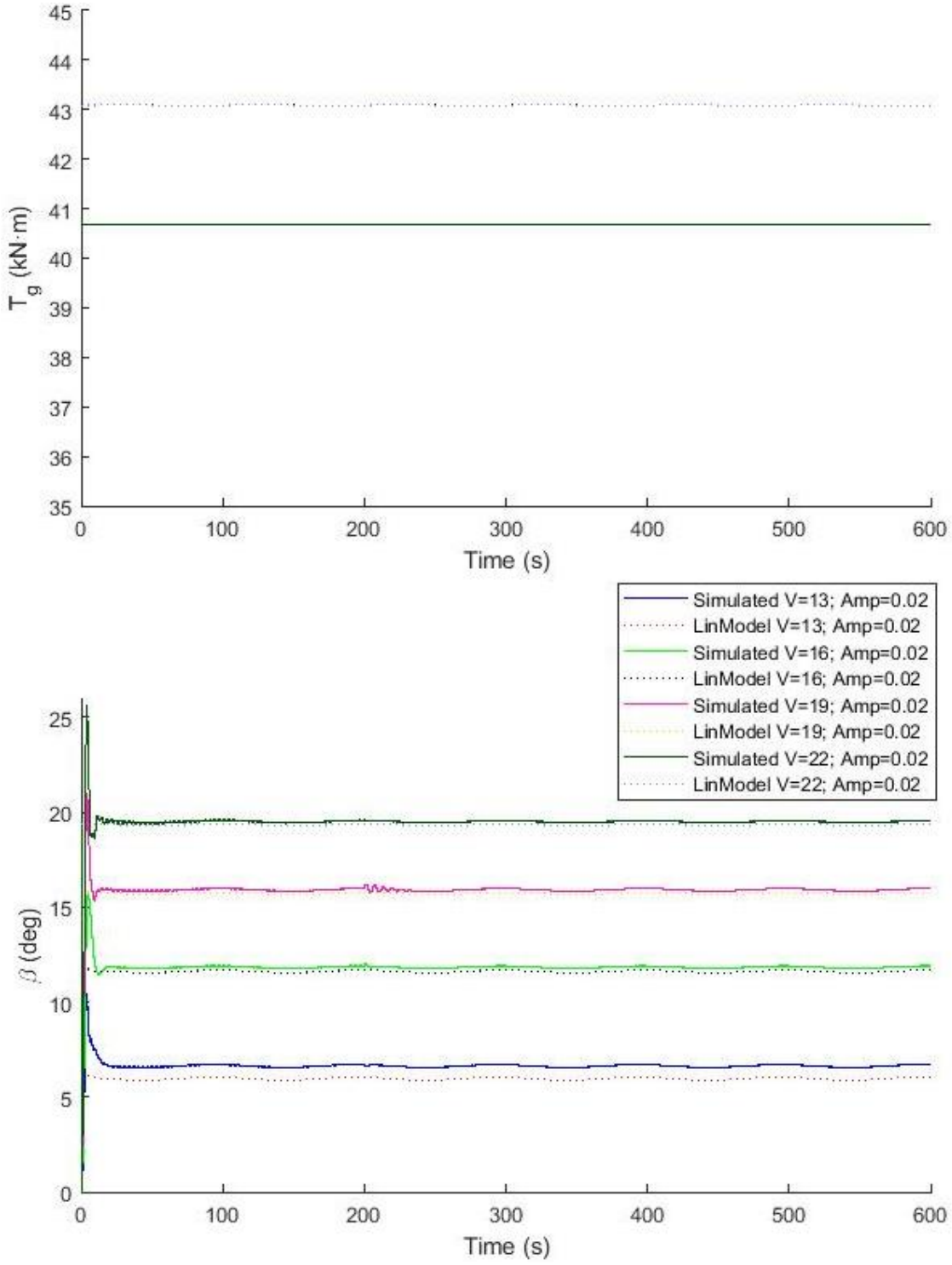

Figura 107. Seno de pitch en lazo cerrado. Tg y β

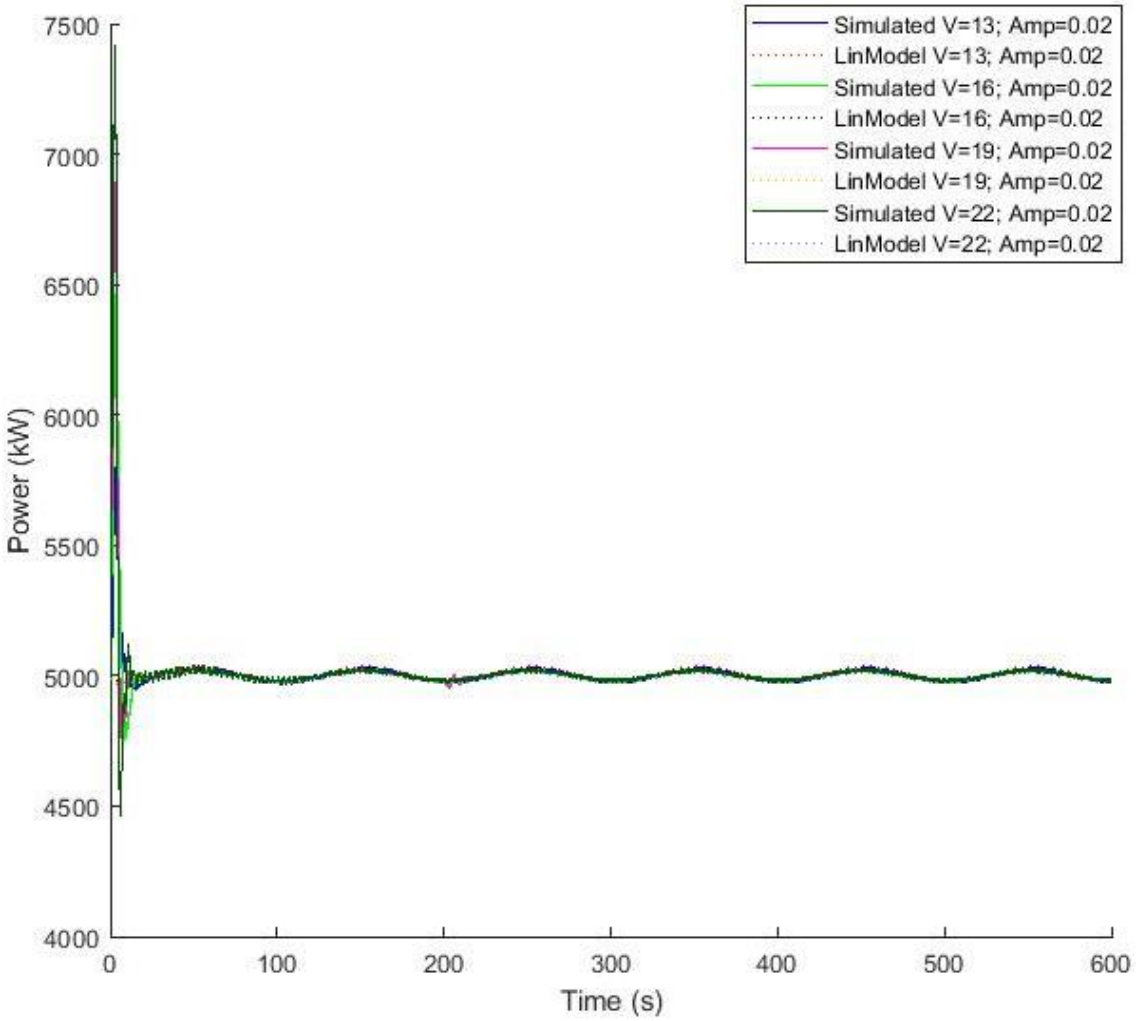

Figura 108. Seno de pitch en lazo cerrado. Potencia

Se introduce una perturbación **senoidal de par** de amplitud 1000 N·m cuando la máquina está trabajando a 4, 6, 8 y 10 m/s (Figura 109). Al corresponderse con velocidades de viento de control de par, el pitch tiene valor nulo (Figura 110).

El comportamiento de ambos modelos es similar, variando el par del generador de salida con una forma de onda senoidal y con una velocidad del generador senoidal. Sin embargo, a 4 m/s, el modelo lineal se desestabiliza (Figura 109) debido a la proporcionalidad de la amplitud del seno en comparación con las otras velocidades de viento y su par de equilibrio (Figura 110).

Tanto en la velocidad del generador (Figura 109) como en la potencia extraída (Figura 111) se observa un offset entre los puntos de equilibrio de ambos modelos. Sin embargo, la dinámica es similar.

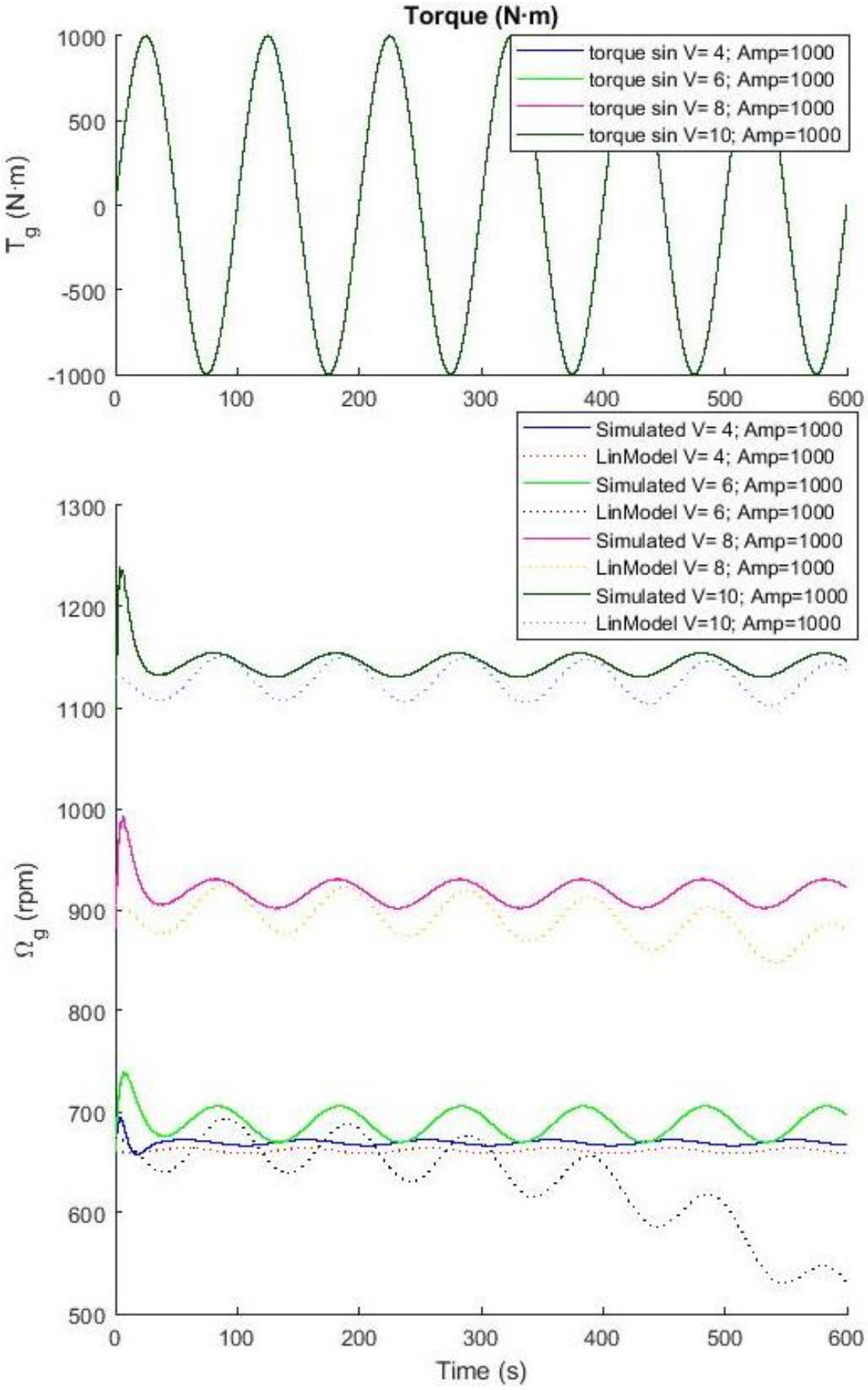

Figura 109. Seno de par en lazo cerrado. Ω_g

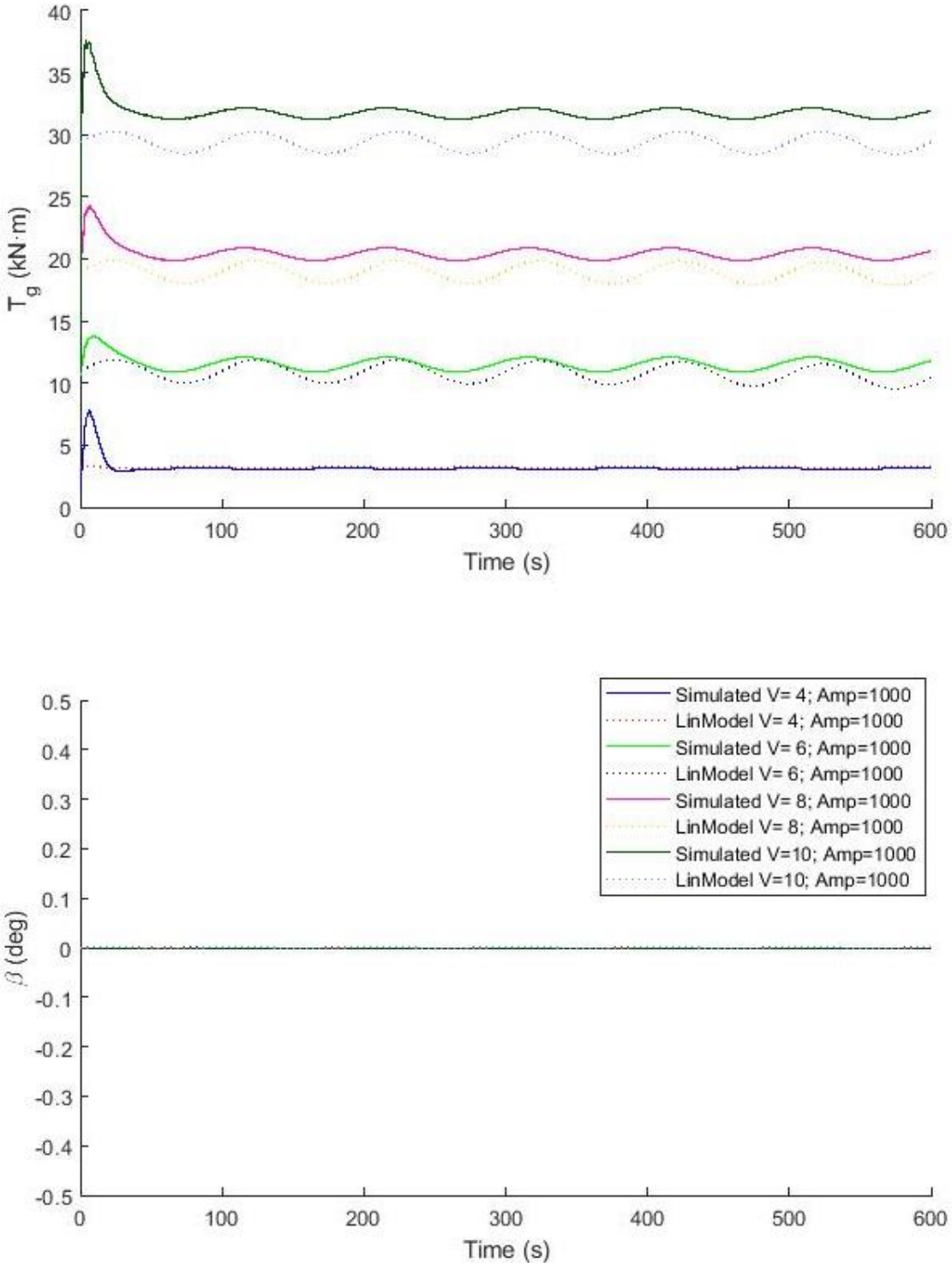

Figura 110. Seno de par en lazo cerrado. Tg y β

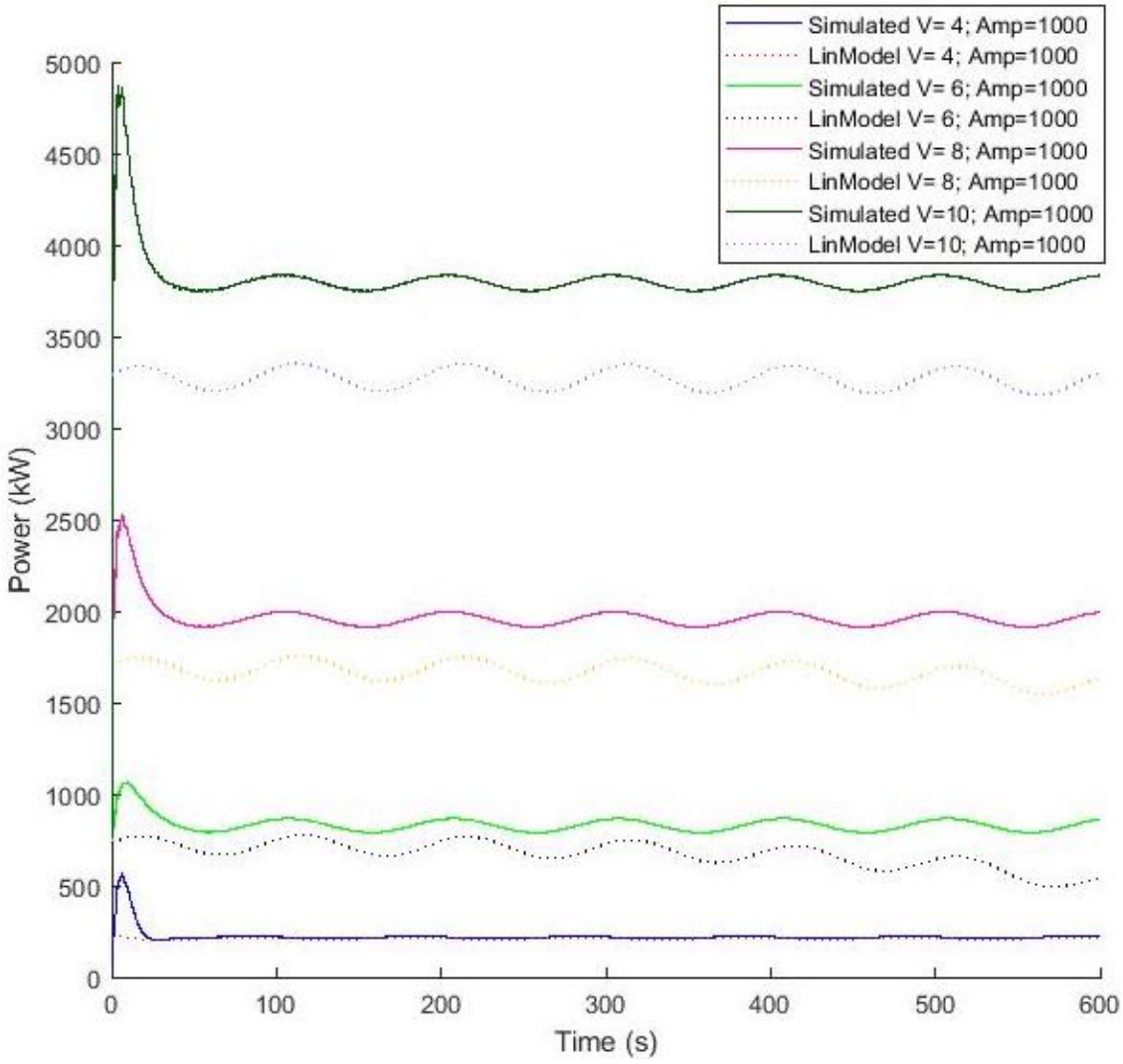

Figura 111. Seno de par en lazo cerrado. Potencia

Completo

Con el objetivo de caracterizar la máquina en todo el rango de velocidades de viento, se introduce primero una **rampa ascendente** desde 3 m/s hasta 25 m/s (Figura 112). El modelo lineal no se asemeja al modelo no lineal porque emplea el modelo de la turbina correspondiente a 3 m/s por lo que trabaja en la primera vertical durante toda la simulación.

A partir de los resultados obtenidos con el modelo no lineal, se observa que la velocidad de giro del generador inicial corresponde con la mínima del generador, pero una vez alcanza la zona cuadrática, esta aumenta hasta llegar y mantenerse a la velocidad nominal (Figura 112).

En cuanto al par, al comienzo de la simulación este aumenta hasta alcanzar su valor nominal, mientras que el ángulo de pitch comienza siendo nulo, para comenzar a crecer una vez el par llega a nominal (Figura 113). Como consecuencia de la relación entre velocidad de giro del generador, el par y el pitch, la potencia aumenta hasta alcanzar su valor nominal (5 MW) una vez la turbina trabaja en zona nominal (Figura 114).

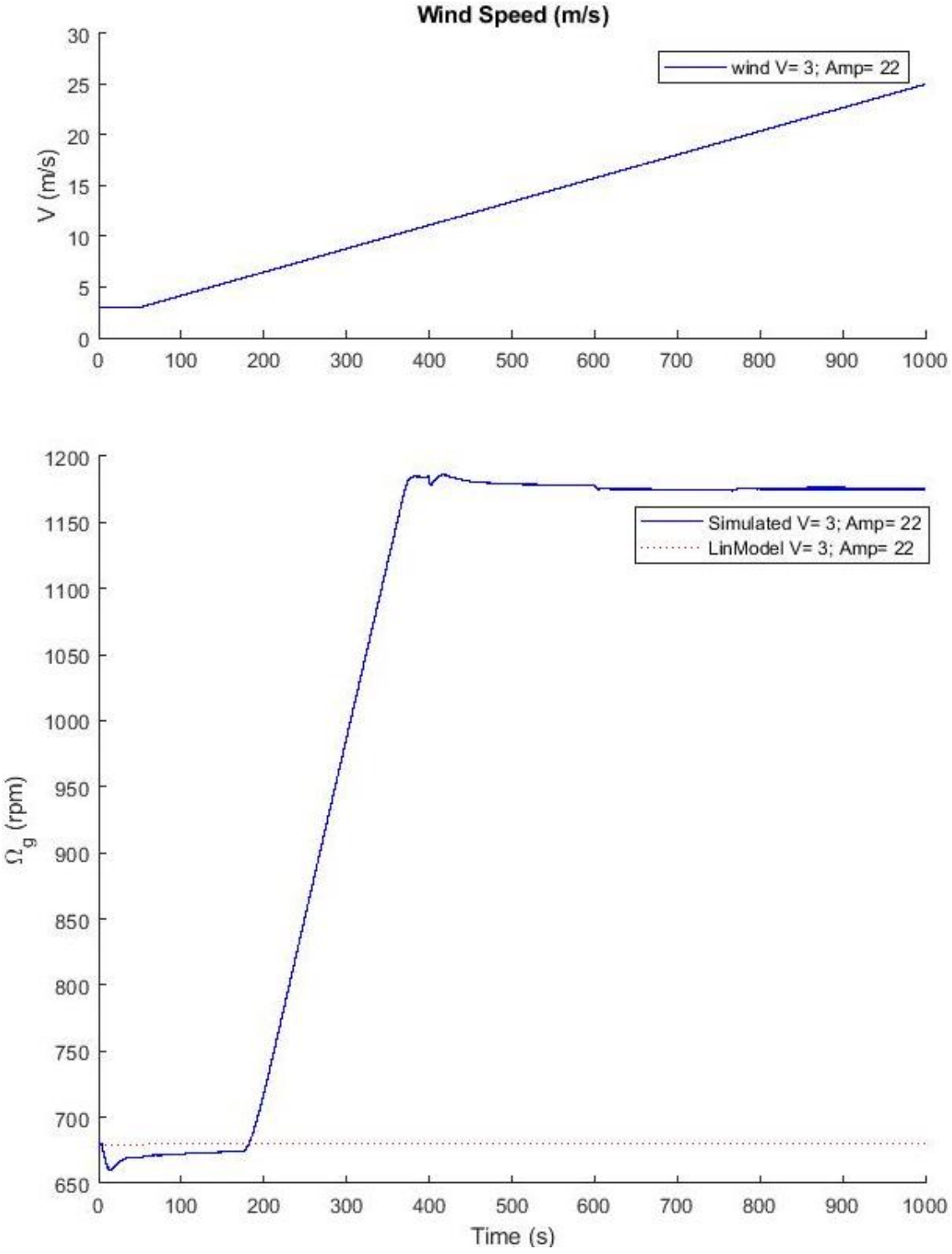

Figura 112. Rampa completa ascendente de viento en lazo cerrado. Ω_g

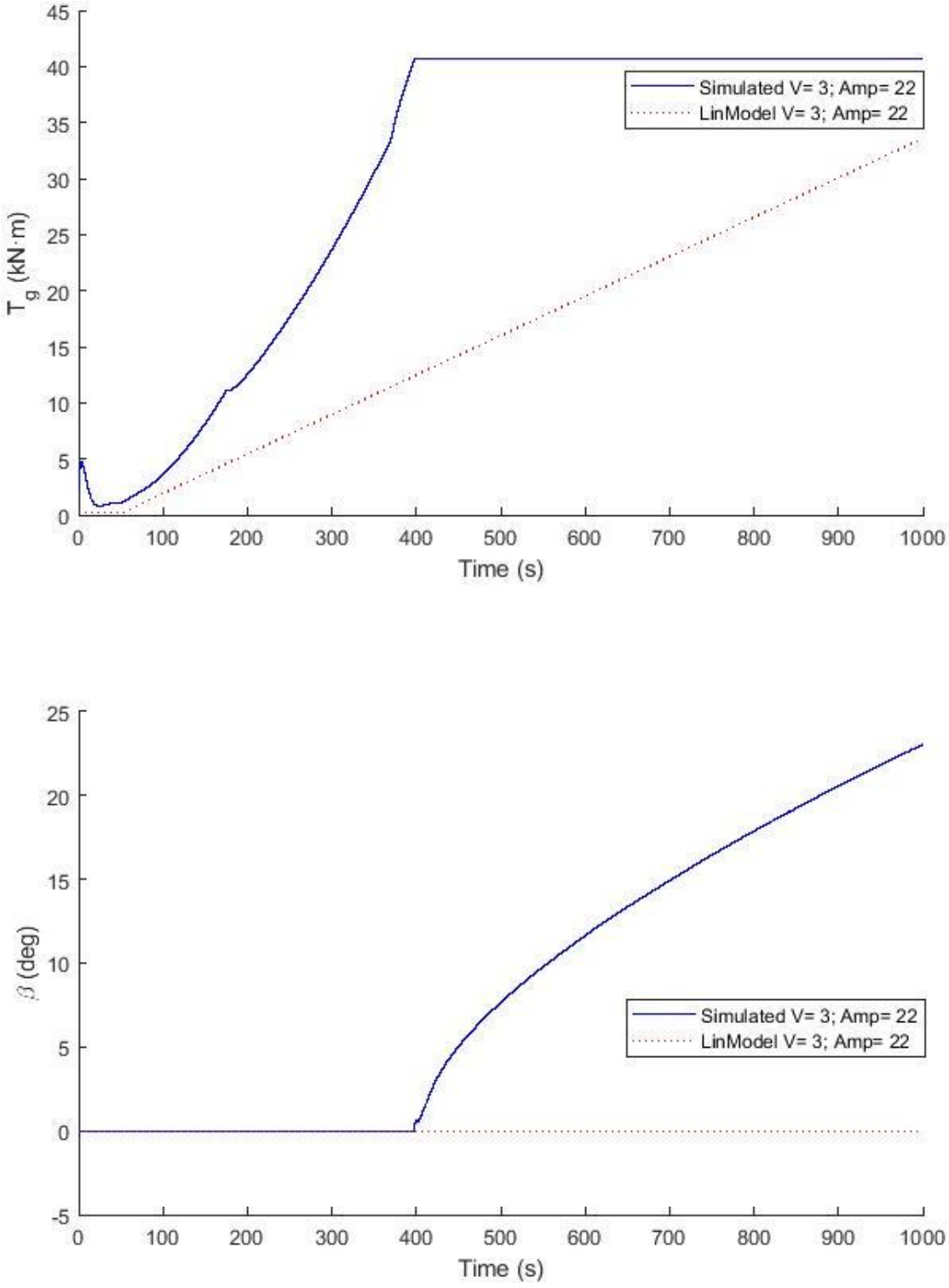

Figura 113. Rampa completa ascendente de viento en lazo cerrado. Tg y β

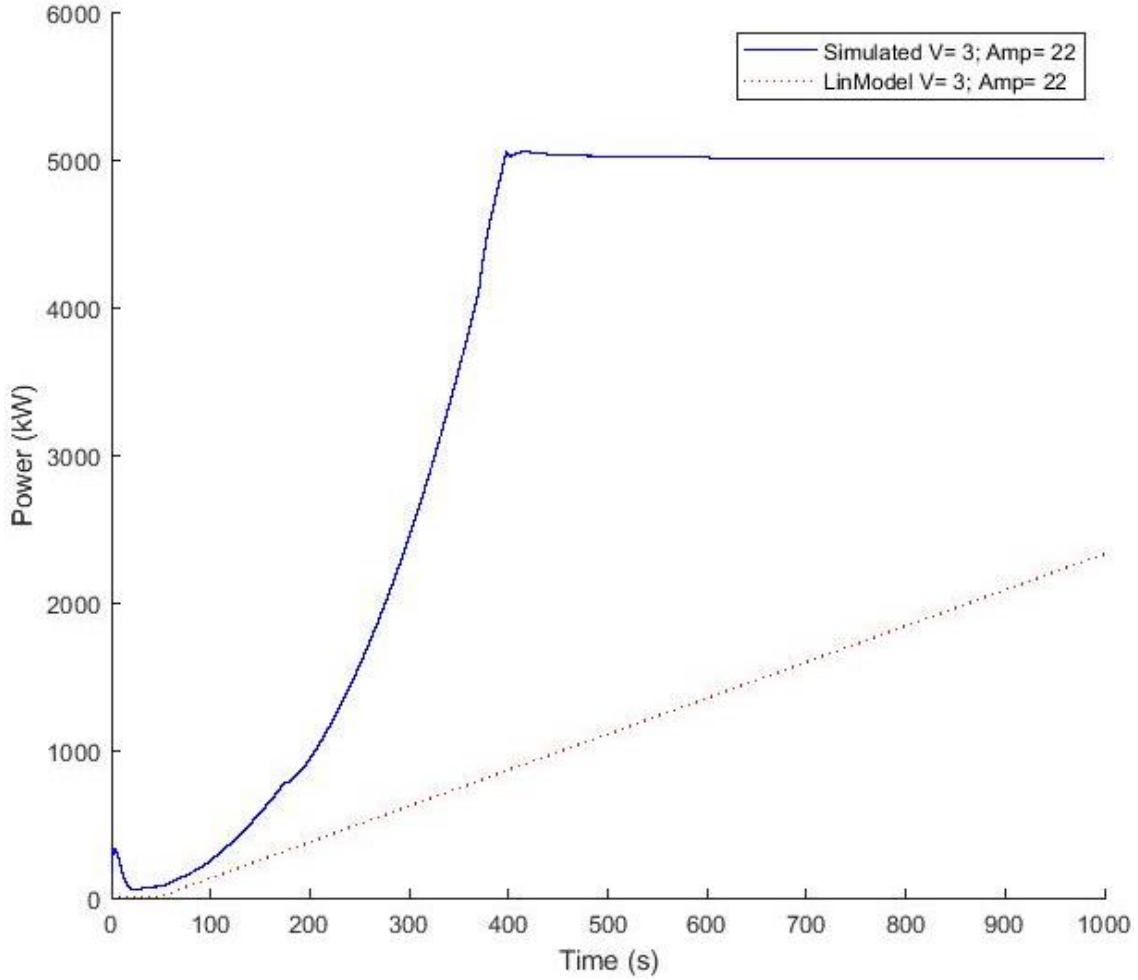

Figura 114. Rampa completa ascendente de viento en lazo cerrado. Potencia

En la Figura 115 se resume el comportamiento del modelo no lineal de la turbina eólica con las zonas identificadas por la máquina de estados empleada en la implementación del lazo cerrado en Simulink.

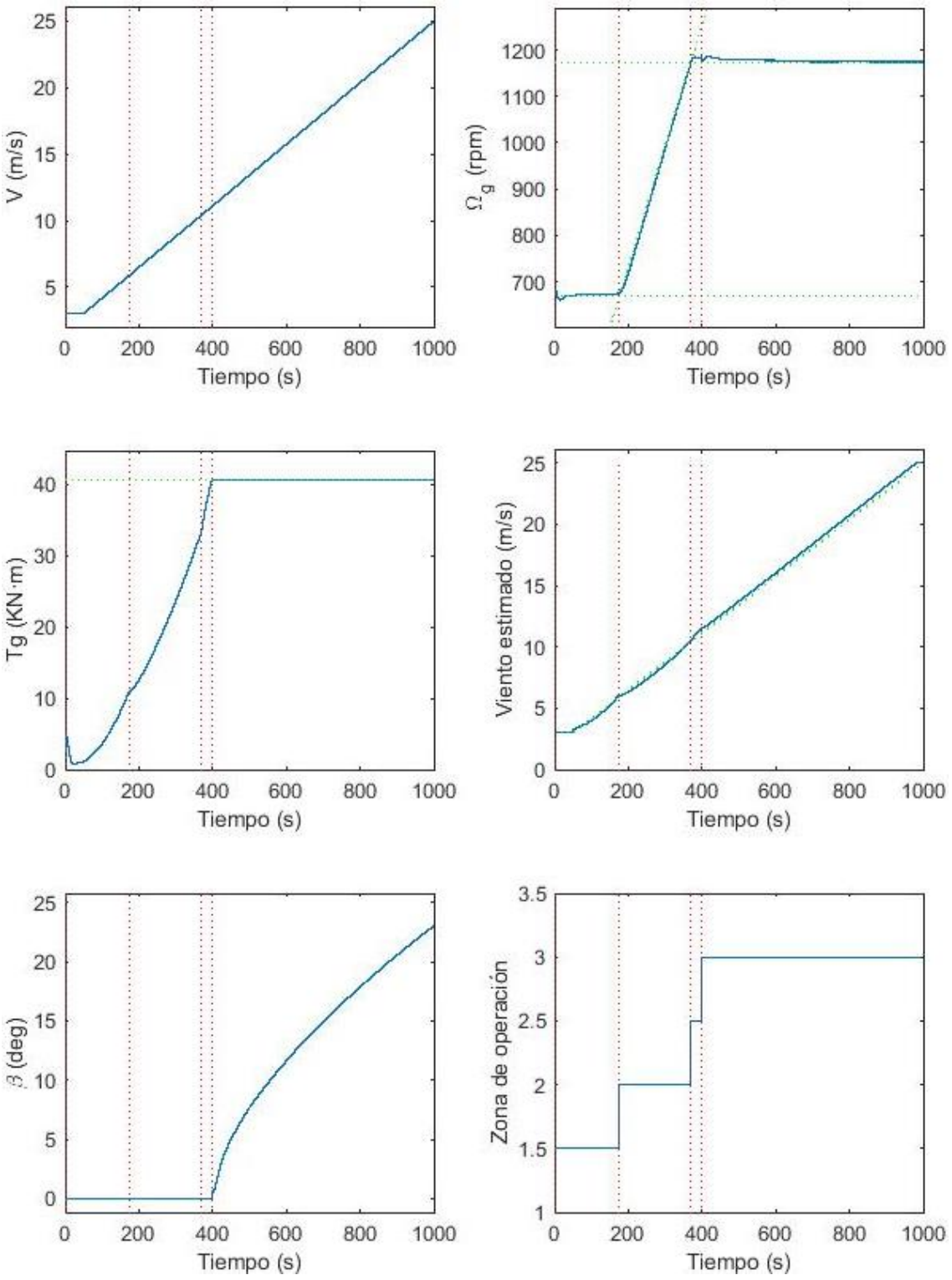

Figura 115. Rampa completa ascendente de viento en lazo cerrado. Comportamiento general

En la Figura 116 se representan los resultados obtenidos de par generador y velocidad de giro del generador con los límites teóricos de la máquina. El comportamiento obtenido se asemeja en gran medida al esperado según descrito en el apartado Zonas de operación (Figura 11).

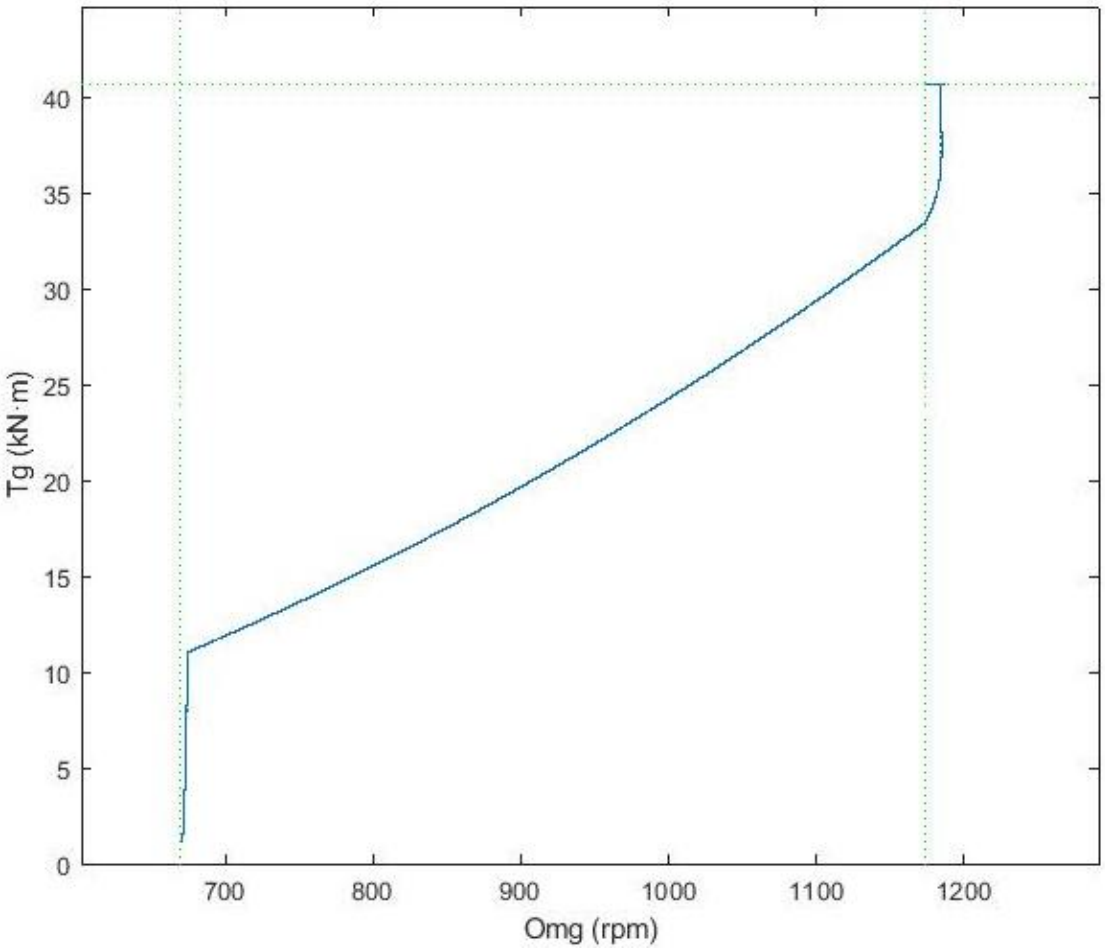

Figura 116. Rampa completa ascendente de viento en lazo cerrado. Curva Tg - Ω_g

Para completar la validación del sistema, se realiza una **rampa descendiente** de viento de 25 m/s hasta 3 m/s (Figura 117). Al igual que ocurría en la rampa ascendente, el modelo lineal no se parece al modelo no lineal debido a que emplea el modelo de la turbina de 25 m/s a lo largo de toda la simulación. Por ello, el modelo lineal trabaja en zona nominal, manteniendo el par constante y realizando control de pitch para todas las velocidades de viento (Figura 118). Esto no indica que el modelo lineal sea erróneo, sino que el modelo de la máquina 25 m/s no es capaz de representar la máquina a todas las velocidades de viento.

A partir de los resultados obtenidos con el modelo no lineal, se observa que la velocidad de giro del generador inicial corresponde con la nominal del generador, pero una vez alcanza la zona cuadrática, esta desciende hasta llegar y mantenerse a la velocidad mínima (Figura 117).

En cuanto al pitch, al comienzo de la simulación, tras un pequeño transitorio, este desciende hasta alcanzar el valor nulo. Mientras el pitch tiene valor mayor que 0, el par se mantiene constante, pero una vez se desactiva el control de pitch, el control de par entra en funcionamiento siguiendo la segunda vertical, la zona cuadrática y la primera vertical (Figura 118). Como consecuencia de la relación entre velocidad de giro del generador, el par y el pitch, la potencia comienza a su valor nominal para luego ir disminuyendo conforme la velocidad de viento disminuye (Figura 119).

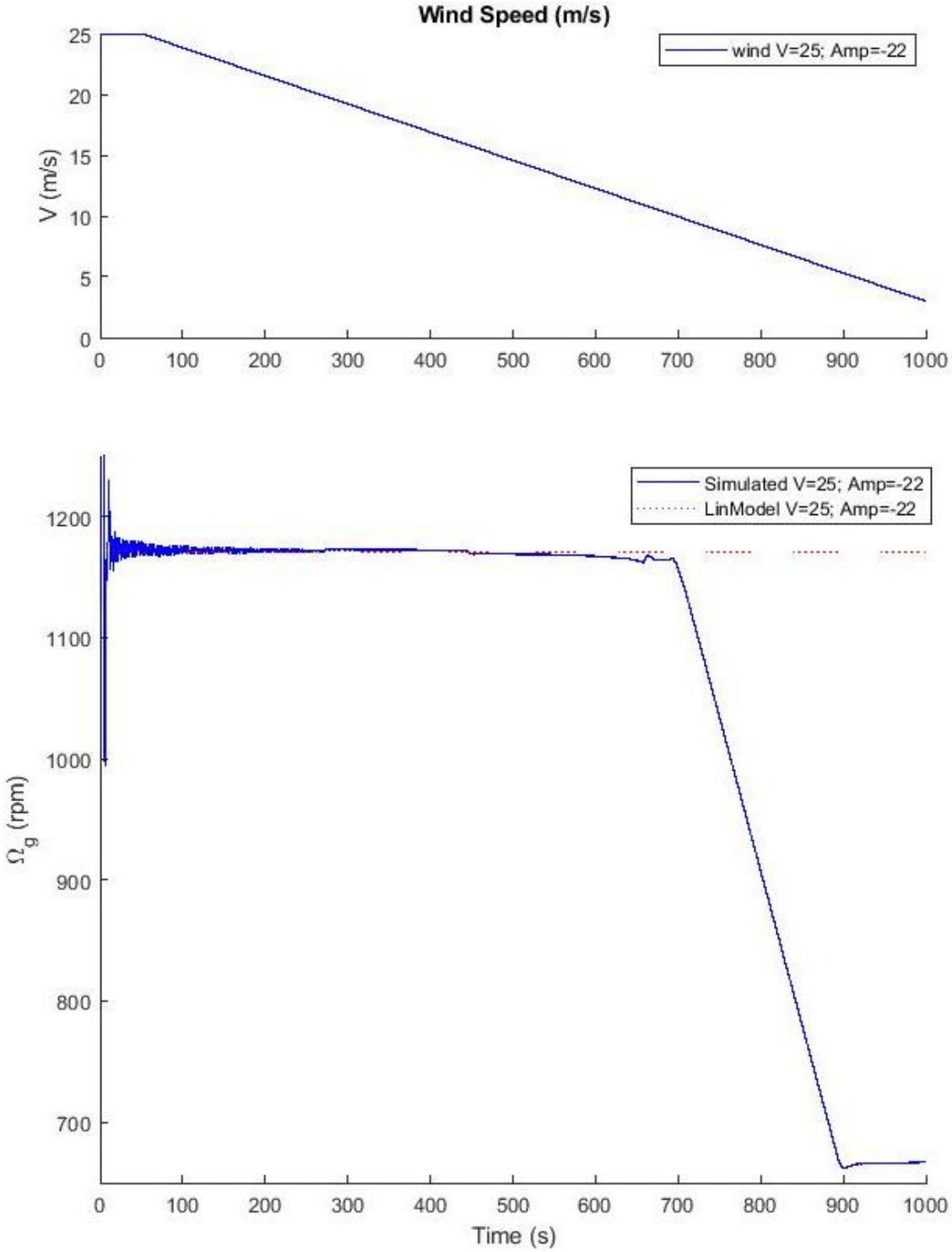

Figura 117. Rampa completa descendente de viento en lazo cerrado. Ω_g

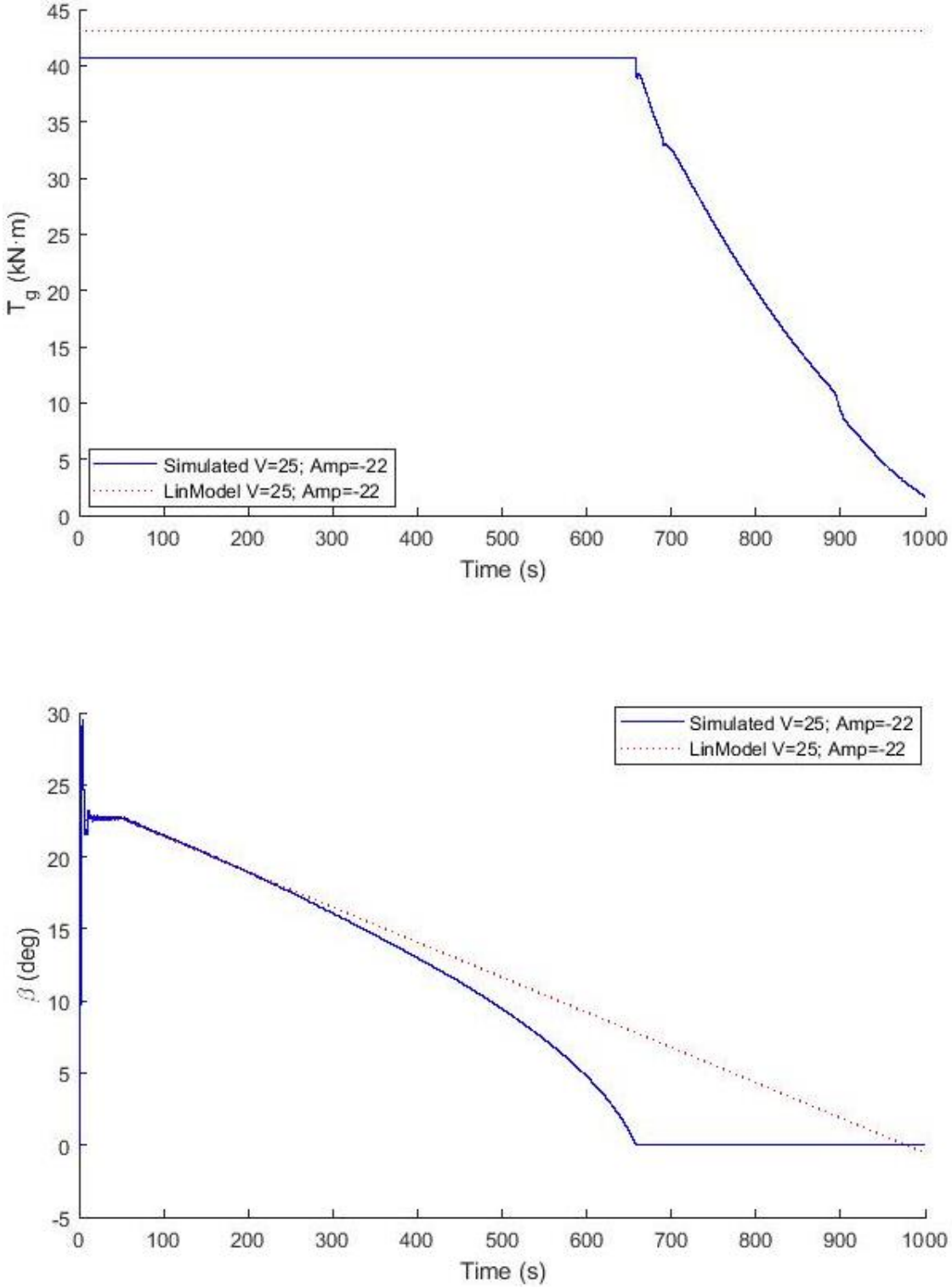

Figura 118. Rampa completa descendente de viento en lazo cerrado. Tg y β

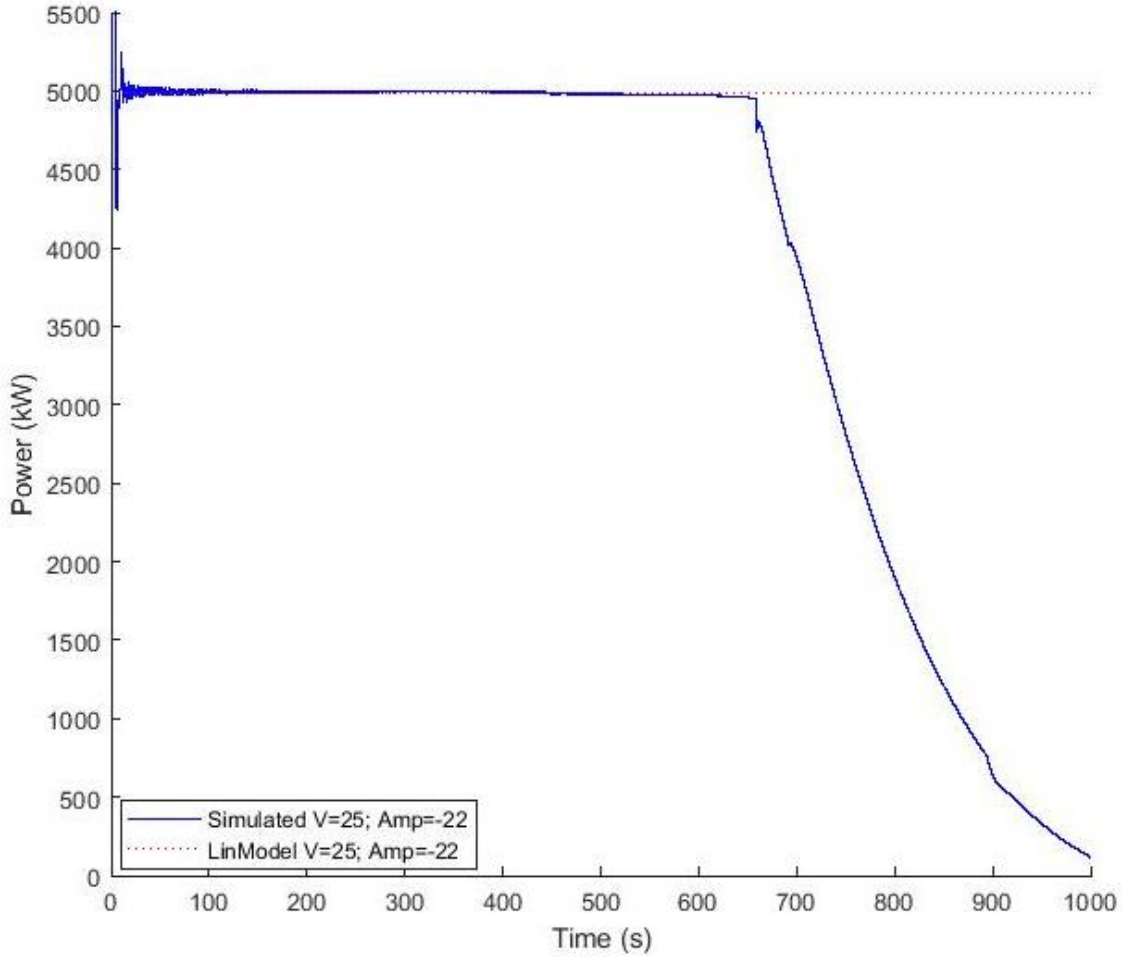

Figura 119. Rampa completa descendente de viento en lazo cerrado. Potencia

En la Figura 120 se resume el comportamiento del modelo no lineal de la turbina eólica ante una rampa descendente de viento con las zonas identificadas por la máquina de estados empleada en la implementación del lazo cerrado en Simulink

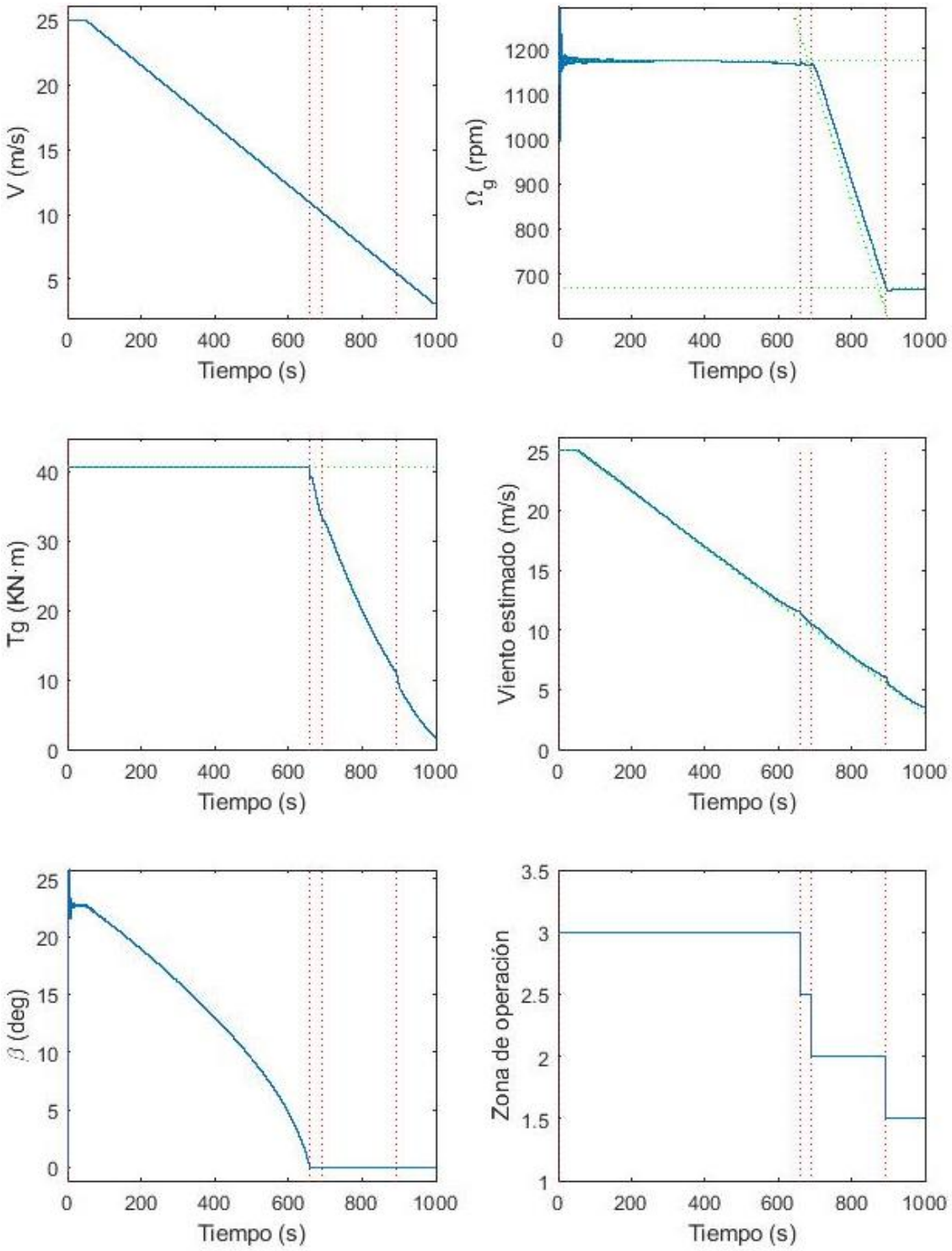

Figura 120. Rampa completa descendente de viento en lazo cerrado. Comportamiento general

En la Figura 121 se representan los resultados obtenidos de par generador y velocidad de giro del generador con los límites teóricos de la máquina. El comportamiento obtenido se asemeja en gran medida al esperado según descrito en el apartado Zonas de operación (Figura 11) y al obtenido en el caso de rampa ascendente (Figura 116).

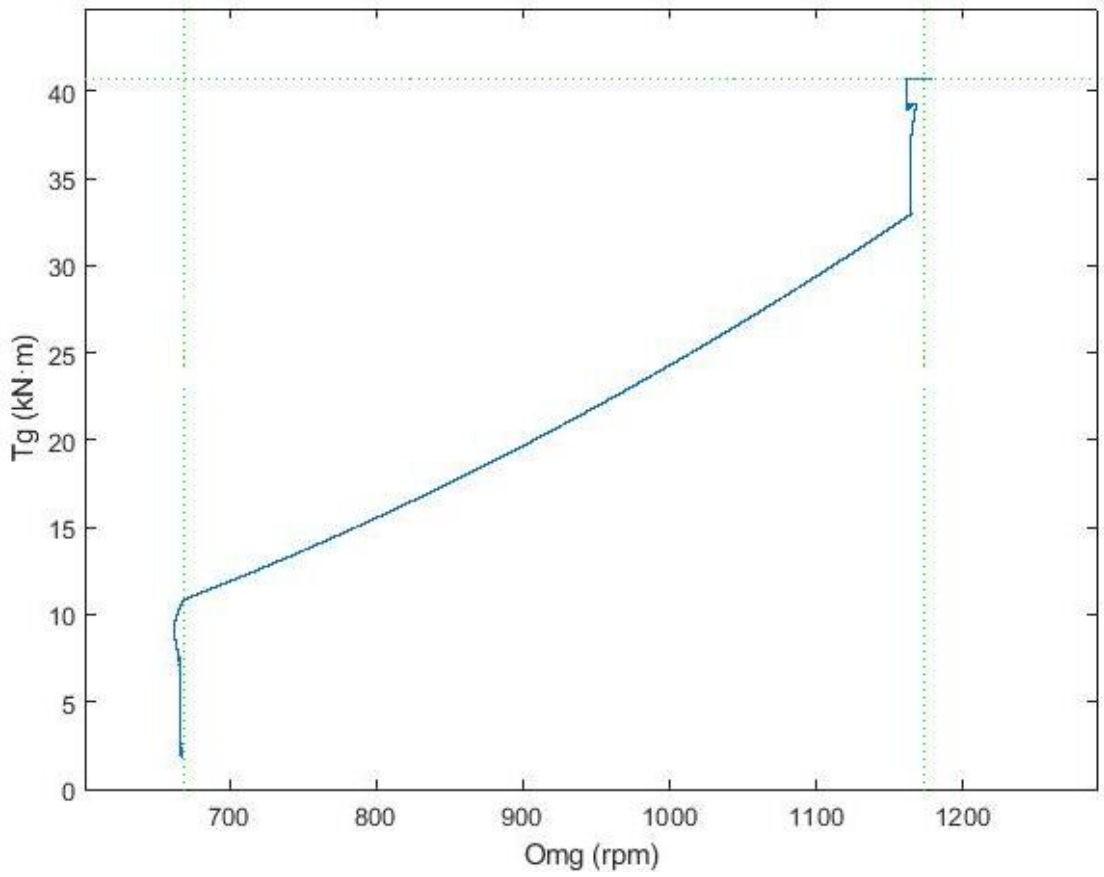

Figura 121. Rampa completa descendente de viento en lazo cerrado. Curva Tg - Ω_g

Análisis de los resultados

A partir de las simulaciones en lazo abierto y lazo cerrado se observa que la dinámica del modelo lineal y del modelo real de FAST es similar. En cuanto a los puntos de equilibrio, hay pequeñas diferencias, tanto en lazo abierto como en lazo cerrado. Sin embargo, al ser el principal uso del modelo lineal, el diseño del control del sistema, la dinámica así como el análisis frecuencial tienen mayor importancia que la exactitud de los puntos de equilibrio.

Las mayores diferencias observadas se deben a que el modelo lineal sólo usa el modelo de la turbina a una velocidad de viento, mientras que el modelo no lineal o general integra el comportamiento del aerogenerador a todas las velocidades de viento. Por ello, cuanta mayor variación de la entrada o perturbación, mayor será la diferencia entre ambos modelos (ejemplo: Figura 112).

En la siguiente tabla se recoge un pequeño resumen del análisis de cada validación realizada.

Tipo		Entrada	Análisis
Lazo abierto	Escalón	Viento	En el escalón de 4 a 5 m/s el modelo lineal se acelera rápidamente debido a que el escalón es porcentualmente mayor y se emplea el mismo modelo de la turbina a 4 m/s que a 5 m/s. A 7 y 11 m/s hay diferencias notables en el punto de equilibrio pero la dinámica es similar.
		Pitch	Hay diferencias en el punto de equilibrio pero la dinámica justo tras el escalón es similar.
		Par	En los escalones a 4 m/s y a 6 m/s el modelo lineal se tiende a frenarse más rápido debido a que una diferencia de 1000 N de par es porcentualmente mayor. Hay diferencias en el punto de equilibrio a 6, 8 y 10 m/s pero la dinámica tras el escalón es similar.
	Rampa	Viento	Hay diferencias en el punto de equilibrio pero la dinámica durante la rampa es similar. En la rampa de 4 m/s a 7 m/s el modelo lineal se acelera rápidamente y el real no sigue el patrón de las otras rampas. Esto se debe a que cuanto menor es la velocidad de viento inicial, más se acelera la máquina al introducirle la misma rampa.
		Pitch	Hay pequeñas diferencias en el punto de equilibrio y la dinámica es similar. A 13 m/s el sistema tiende a reducir su velocidad de giro más rápido al ser proporcionalmente una rampa de pitch mayor.
		Par	Hay diferencias en el punto de equilibrio pero la dinámica inicial es similar. Cuanto menor es la velocidad de viento, antes difieren el modelo lineal del real al ser proporcionalmente una rampa mayor.
	Seno	Viento	La dinámica del sistema es similar en ambos casos con un desvío sobre el valor medio debido al punto de equilibrio. El modelo lineal a 4 m/s no sigue la forma senoidal, sino que se acelera.
		Pitch	La dinámica del sistema es similar y el desvío sobre el valor medio senoidal es pequeño.
		Par	La dinámica del sistema es similar salvo a 4 m/s y 6 m/s que el modelo lineal reduce su velocidad de giro en vez de seguir la forma senoidal esperada. Hay diferencias en el punto de equilibrio que se traducen en una variación del valor medio senoidal.
Diagrama de Bode		Viento	Ambos diagramas de Bode se asemejan mucho. Para obtener mejores resultados habría que aumentar el número de muestras a partir de 0.5 Hz y aumentar el tiempo de simulación para poder obtener los valores de fase y ganancia con el sistema más estable.

		Pitch	Los resultados son similares para el modelo lineal y el real o simulado. Las mayores diferencias se obtienen en la fase. Para el Bode Pitch - Pitch, el resultado es el esperado: constante con ganancia 35.16 y fase 0.
		Par	Los resultados son similares obteniendo muy buenos resultados en las frecuencias más altas analizadas pero con diferencias en fase y en la ganancia del diagrama de potencia. Para el Bode Par – Par, el resultado es el esperado: constante con ganancia -60 y fase 0.
Lazo cerrado	Escalón	Viento	Comportamiento similar. A 7 m/s difiere el modelo lineal del real debido al punto de equilibrio, pero la dinámica sigue siendo semejante.
		Pitch	Comportamiento similar. En ambos casos se amortigua el escalón de pitch
		Par	A 8 y 10 m/s, antes de introducir el escalón se observa diferencias notables en cuanto al punto de equilibrio. Al introducir el escalón, el modelo lineal tiende a reducir la velocidad de la máquina; mientras que en el modelo real, si se mantiene en zona 1.5, la velocidad sigue siendo la mínima, pero si pasa a zona 2, la máquina se acelera sin superar la velocidad nominal.
	Rampa	Viento	Cuando la rampa se produce en zona 3, el comportamiento es similar. Al introducirlo en zona 2, el modelo lineal tiende a acelerarse más debido a que se emplea el modelo del aerogenerador a dicha velocidad de viento. En la rampa a 4 m/s se observa al final de la simulación el cambio de zona de 1.5 a 2 (la máquina se acelera). En la rampa a 11 m/s se observa como el modelo lineal sigue trabajando en zona 2-2.5 mientras que el simulado realiza control de pitch (zona 3).
		Pitch	Comportamiento similar. Tanto el modelo lineal como el simulado trabajan con control de pitch con valores parecidos.
		Par	Al introducir la rampa de par en el modelo lineal, la máquina se frena. Sin embargo, en el simulado, el control se opone a dicha perturbación de par, manteniendo la velocidad de giro constante al supuesto valor correspondiente a cada velocidad de viento para maximizar la potencia extraída.
	Seno	Viento	La dinámica es similar. En el caso del modelo lineal, la variación de velocidad en la zona 2 es mayor al tener una menor variación del control de par. En zona 3 el comportamiento es semejante. Hay que tener cuidado a la hora de introducir señales sinusoidales en torno a 11 m/s.
		Pitch	Comportamiento similar. Tanto el modelo lineal como el simulado trabajan con control de pitch con valores parecidos.

		Par	Comportamiento similar. La mayor diferencia se encuentra en el modelo lineal a 4 m/s que tiene a reducir su velocidad por debajo de la mínima.
	Rampa completa	Ascendente	Al introducir una rampa de gran amplitud (22 m/s), el modelo lineal de la turbina que se emplea no es válido para ese rango de velocidades de viento, por lo tanto, es de esperar que ambos modelos no sean similares. Se observa que el comportamiento del modelo no lineal se asemeja al estudiado en el apartado Fundamentos teóricos a partir del modelo matemático del aerogenerador.
		Descendente	

Tabla 12. Análisis validación

Conclusiones

Para la validación de la herramienta de linealización de FAST, se ha creado un procedimiento basado en scripts de MATLAB que compara los resultados del modelo lineal con el modelo no lineal simulado en FAST. Dicho procedimiento pasa por el análisis tanto temporal como frecuencial en lazo abierto y en lazo cerrado, no solo permitiendo la validación de linealizaciones de FAST, sino que también linealizaciones realizadas mediante otras herramientas de linealización.

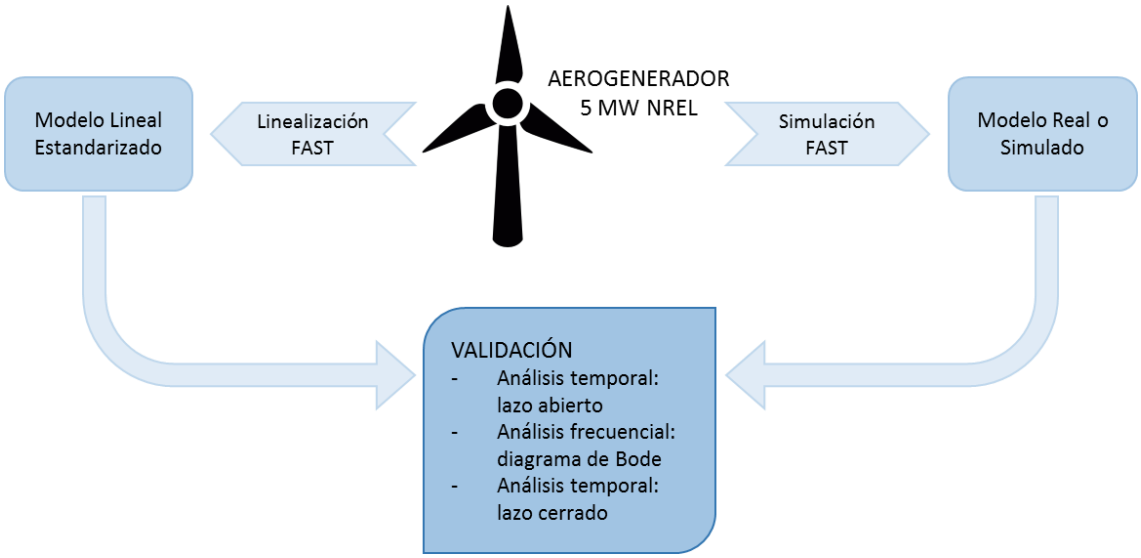

Figura 122. Esquema general

A partir de los resultados obtenidos en la validación, puede concluirse que la herramienta de linealización de FAST es válida para todas las zonas de operación del aerogenerador. Esto permite diseñar el control de cada una de las zonas basándose en dicho modelo lineal.

Como aportación innovadora frente a otros proyectos, se ha linealizado y validado la linealización del aerogenerador a bajas velocidades de viento, permitiendo el estudio de otros posibles controladores o mejoras en la zona cuadrática del aerogenerador.

Además, se ha creado una herramienta post-procesadora de FAST para obtener la linealización siguiendo una estructura estandarizada similar a la empleada en la obtención de modelos lineales con el software comercial BLADED. Esto permite la compatibilidad de las linealizaciones obtenidas con FAST con post-procesadores comerciales empleados en el sector eólico.

Nombre	Descripción
lin_FAST	Herramienta para la linealización de un vector de vientos con FAST y su post-procesado con la finalidad de obtener un modelo lineal de acuerdo con la estandarización empleada a nivel comercial.
OpenLoopValidation	Herramienta para la validación temporal en lazo abierto de un modelo lineal. Permite comparar el sistema real simulado con FAST con un modelo lineal en lazo abierto aplicándole escalones, rampas y senos de viento, pitch o par tanto de amplitud positiva como negativa.

BodeValidation	Herramienta para la validación frecuencial en lazo abierto de un modelo lineal. Permite comparar el sistema real simulado con FAST con un modelo lineal en lazo abierto en términos frecuenciales empleando el diagrama de Bode. Para ello, al sistema real se le aplican senos de viento, pitch o par a diferentes frecuencias.
ClosedLoopValidation	Herramienta para la validación temporal en lazo cerrado de un modelo lineal. Permite comparar el sistema real simulado con FAST con un modelo lineal en lazo cerrado aplicándole escalones, rampas y senos de viento, perturbaciones de pitch o perturbaciones de par tanto de amplitud positiva como negativa. Para el cierre de lazos del modelo no lineal se emplea Simulink con una estructura de controladores suavizada y con un estimador de la velocidad de viento.

Tabla 13. Descripción de las principales herramientas desarrolladas

Bibliografía

[1] P. J. López Bracot, "Desarrollo de un entorno de validación profesional que facilite el diseño fiable de estrucutras de control y estimadores de cargas en aerogeneradores," Universidad Pública de Navarra; IMAC, 2018.

[2] M. Bontigui Vallejos, "Impact of generator speed filtering process on wind turbine structural loads," Universidad Pública de Navarra, 2019.

[3] J. Á. Álvarez Echarri, "Estudio De Diferentes Topologías De Generación Eólica," 2011.

[4] NREL, "NWTC Information Portal," 2019. [Online]. Available: https://nwtc.nrel.gov/.

[5] J. Jonkman, S. Butterfield, W. Musial, and G. Scott, "Definition of a 5-MW Reference Wind Turbine for Offshore System Development," no. February. 2009.

[6] J. M. Jonkman and M. L. J. Buhl, "FAST User's Guide - Updated August 2005," no. October. 2005.

[7] I. Lizarraga Zubeldia, "Entorno de simulación para verificar diseños basados en linealizaciones," Universidad Pública de Navarra, 2019.

[8] AENOR, "UNE-EN 61400-1:2006 Aerogeneradores. Parte 1: Requisitos de diseño (CEI 61400-1:2005)." p. 96, 2006.

ANEXO A: Linealización con FAST

```
clear all
clc
close all

cd D:\IDOIA\TFM\FAST
```

Indicaciones de requisitos de los diferentes archivos para evitar errores:

```
% Comprobar que el archivo .ipt tiene bien escrito la dirección del DETFILE
% y los .dat aerodinámicos (cylinder...)

% Comprobar que el archivo .fst empleado está en modo create a periodic
% linearized model AnalMode = 2 (8).
% Comprobar que el archivo .fst tiene bien escrito la dirección del .dat de
% la torre (135), del .dat de las palas (157-159), del .itp (161) y del
% .dat de linealización (167).
% Ordenar las salidas del .fst según BLADED:
% "RotSpeed"                          - Rotor velocity (rpm)
% "GenSpeed"                          - Generator velocity (rpm)
% "GenTq"                             - Generator torque (kN·m)
% "BlPitch1"                          - Blade pitch angle (deg)
% "TipSpdRat"                         - Tip Speed Ratio
% "NcIMUTVxs"                         - Nacelle-x-velocity (m/s)
% "NcIMUTAxs"                                    - Nacelle-x-acceleration
(m^2/s)
% "RootMyb1"                          - Blade 1 flapwise moment My (kN·m)
% "RootMxb1"                          - Blade 1 edgewise moment Mx (kN·m)
% "LSShftMxa"                                  - LSS torque (EQUIVALENT TO THE
ROTOR TORQUE)
% "TwrBsMyt"                          - Tower base pitching torque (kN·m)
% "GenPwr"                            - Generator Power (kW)

% Comprobar que en run.bat las direcciones de FAST_exe (FAST) y
% DateTime.exe (DateTime) están bien escritas, así como el nombre del
% archivo (TESTFILE)
```

Input data

```
Wind_array=3:1:25;  % Wind Speed vector [m/s]
FOLDER = 'D:\IDOIA\TFM\FAST\Cases\linearization\LinModel';
DETfile = 'D:\IDOIA\TFM\FAST\wind\deterministic\wind_det.wnd';
FSTfile =
'D:\IDOIA\TFM\FAST\Cases\linearization\LinModel\linearization_gen.fst';
LINfile =
'D:\IDOIA\TFM\FAST\Cases\linearization\LinModel\linearization_gen.lin';
DATfile =
'D:\IDOIA\TFM\FAST\Cases\linearization\LinModel\Param_linearization.dat';
AERfile = 'D:\IDOIA\TFM\FAST\Cases\linearization\LinModel\aerodyn_gen.ipt';
DISfile = 'D:\IDOIA\TFM\FAST\Cases\linearization\LinModel\DISCON.in';

Jr = 35444067;                      % Rotor inertia
load('power_curves_NREL5MW_v2.mat') % betas in deg
Omr_nom = 12.1;                     % rpm
Omr_min = 6.9;                      % rpm
Pw_nom = 5e6;                       % W
```

```
lambda_opt = 7.69;                  % -
beta_Z2 = 0;                        % beta [deg] in zone 1.5, 2 and 2.5

% Simulation conditions
t_p_lin = 0.02;     % Integration time step for linearization[s]
t_end_lin = 600;    % Total time for linearization [s]
missalignment = 8;  % By default
```

Obtaining information from the input data

```
% Read WTG parameters from FSTfile
parname{1} = sprintf('%s',' TipRad ');
parname{2} = sprintf('%s',' GenIner ');
parname{3} = sprintf('%s',' GBRatio ');
parname{4} = sprintf('%s',' TowerHt ');
parname{5} = sprintf('%s',' NumBl ');
parname{6} = sprintf('%s',' TMax ');
parname{7} = sprintf('%s',' DT ');
parname{8} = sprintf('%s',' BlPitch(1) ');
parname{9} = sprintf('%s',' BlPitch(2) ');
parname{10} = sprintf('%s',' BlPitch(3) ');
parname{11} = sprintf('%s',' RotSpeed ');
fst = textread(FSTfile,'%s','delimiter','\n');
cont = 0;
parindex = zeros(1,size(parname,2));
parline = cell(1,5);
par = zeros(1,5);
index_fst = zeros(1,6);
for j=1:length(parname)
    parindex(j)=find(~cellfun(@isempty,strfind(fst,parname{j})));
    if j<6
        parline{j}=cell2mat(fst(parindex(j)));
        par(j)=eval(parline{j}(1:strfind(parline{j},parname{j})-1));
    else
        cont=cont+1;
        index_fst(cont)=parindex(j);
    end
end
R = par(1);             % Wind turbine radio [m]
Jg = par(2);            % Generator inertia
Gbx = par(3);           % Gearbox ratio
Jtot = Jr + Gbx^2*Jg;   % Total inertia
htower = par(4);        % Tower height [m]
NBlades = par(5);       % Number of blades
clear fst parindex parline par j parname cont

% Read conditions from AERfile
parname{1}=sprintf('%s','Rho');
aer = textread(AERfile,'%s','delimiter','\n');
parindex = zeros(1,size(parname,2));
parline = cell(1,size(parname,2));
par = zeros(1,size(parname,2));
for j=1:length(parname)
    parindex(j)=find(~cellfun(@isempty,strfind(aer,parname{j})));
    parline{j}=cell2mat(aer(parindex(j)));
    par(j)=eval(parline{j}(1:strfind(parline{j},parname{j})-1));
end
rho = par(1);             % Air density [kg/m^3]
```

```
clear aer parindex parline par j parname

% Read conditions from DATfile
parname{1}=sprintf('%s','TrimCase');
dat = textread(DATfile,'%s','delimiter','\n');
index_dat = zeros(1,size(parname,2));
for j=1:length(parname)
    index_dat(j)=find(~cellfun(@isempty,strfind(dat,parname{j})));
end
clear dat j parname

% Units conversion
radsrpm = 30/pi;          % From rad/s to rpm
rpmrads = 1/radsrpm;      % From rpm a rad/s
degrad = pi/180;          % From degrees to rad
raddeg = 1/degrad;        % From rad a grados (deg)
% Units used in FAST: [rad], [rpm], [m/s], [m/s^2], [N], [N·m], [W], [s]

Tg_nom = Pw_nom/(Omr_nom*rpmrads*Gbx); % N·m
cp_max = griddata(lambdas,betas,cps',lambda_opt,beta_Z2);
K_opt = 1/(2*Gbx)*rho*pi*R^2*cp_max*(R^3/lambda_opt^3)*rpmrads^2/Gbx^2;
```

FAST Lineatization

```
% Used Variables
Omr_init = zeros(1,length(Wind_array));
Pitch_init = zeros(1,length(Wind_array));
Tg_init = zeros(1,length(Wind_array));
Pw_init = zeros(1,length(Wind_array));
Lambda_init = zeros(1,length(Wind_array));
Cp_init = zeros(1,length(Wind_array));
Region = zeros(1,length(Wind_array));
Type_control=2;

for posV=1:length(Wind_array)

        Omr_init(posV) = Wind_array(posV)*lambda_opt/R*radsrpm; % Rotor Speed
[rpm] when lambda opt

        % REGION 1.5 - First Vertical. Pitch = 0
        if Omr_init(posV) < Omr_min
            Omr_init(posV) = Omr_min;       % Rotor Speed [rpm]
            Type_control=2;
            Pitch_init(posV) = beta_Z2;
            Tg_init(posV) = K_opt*(Omr_init(posV)*Gbx)^2;
            Pw_init(posV) = Tg_init(posV)*Omr_init(posV)*rpmrads*Gbx;
            Lambda_init(posV) = Omr_init(posV)*rpmrads*R/Wind_array(posV);
            Cp_init(posV) =
2*Gbx*Tg_init(posV)*Omr_init(posV)*rpmrads/(rho*pi*R^2*Wind_array(posV)^3);
            Region(posV) = 1.5;

        % REGION 2 - Cuadratic Torque Control. Pitch = 0
        elseif Omr_init(posV) < Omr_nom
            Type_control=2;
            Pitch_init(posV) = beta_Z2;
            Tg_init(posV) = K_opt*(Omr_init(posV)*Gbx)^2;
            Pw_init(posV) = Tg_init(posV)*Omr_init(posV)*rpmrads*Gbx;
            Lambda_init(posV) = Omr_init(posV)*rpmrads*R/Wind_array(posV);
```

```
            Cp_init(posV) =
2*Gbx*Tg_init(posV)*Omr_init(posV)*rpmrads/(rho*pi*R^2*Wind_array(posV)^3);
            Region(posV)=2;

        else    % Omr_init(posV) >= Omr_nom
            Omr_init(posV) = Omr_nom;                % Rotor Speed [rad/s] max
(nom)

            % REGION 2.5 - Torque Control. Pitch = 0. TORQUE MUST BE LOWER
THAN NOMINAL
            if Type_control==2
                Type_control=2;
                Pitch_init(posV)=beta_Z2;
                Tg_init(posV) = K_opt*(Omr_init(posV)*Gbx)^2;
                Pw_init(posV) = Tg_init(posV)*Omr_init(posV)*rpmrads*Gbx;
                Lambda_init(posV) = Omr_init(posV)*rpmrads*R/Wind_array(posV);
                Cp_init(posV) =
2*Gbx*Tg_init(posV)*Omr_init(posV)*rpmrads/(rho*pi*R^2*Wind_array(posV)^3);
                Region(posV)=2.5;

            % REGION 3 - Pitch Control. Tg = Tgnom.
            else
                Type_control=3;
                Pw_init(posV) = Pw_nom;
                Tg_init(posV) = Pw_init(posV)/(Omr_init(posV)*rpmrads*Gbx);
                Lambda_init(posV) = Omr_init(posV)*rpmrads*R/Wind_array(posV);
                Cp_init(posV) =
2*Gbx*Tg_init(posV)*Omr_init(posV)*rpmrads/(rho*pi*R^2*Wind_array(posV)^3);
                cps_lambda0(:,posV) =
griddata(lambdas,betas,cps',Lambda_init(posV),betas);
                Pitch_init(posV) =
interp1(cps_lambda0(:,posV),betas,Cp_init(posV));
                if isnan((Pitch_init(posV))) || Pitch_init(posV)<1e-5
                    Pitch_init(posV)=beta_Z2;
                end
                Region(posV)=3;
            end
        end

        %%%%% MODIFICACIÓN ARCHIVOS FAST %%%%%
        % Create the deterministic wind files
        wind_det(t_end_lin, 0, 0, Wind_array(posV), 0, 0, 0, missalignment,
'lin', DETfile);

        % linearization_gen.fst from Cases/linearization is modified with the
desired time and pitch
        fid = fopen(FSTfile,'r');
        i = 1;
        tline = fgetl(fid);
        A_text{i} = tline;
        while ischar(tline)
            i = i+1;
            tline = fgetl(fid);
            A_text{i} = tline;
        end
        fclose(fid);
        A_text{index_fst(1)} = sprintf('%6.0f %s',t_end_lin,'     TMax
- Total run time (s)');
```

```
        A_text{index_fst(2)} = sprintf('%8.4f %s',t_p_lin,'   DT          -
Integration time step (s)');
        A_text{index_fst(3)} = sprintf('%6.1f %s',Pitch_init(posV),'
BlPitch(1)  - Blade 1 initial pitch (degrees)');
        A_text{index_fst(4)} = sprintf('%6.1f %s',Pitch_init(posV),'
BlPitch(2)  - Blade 2 initial pitch (degrees)');
        A_text{index_fst(5)} = sprintf('%6.1f %s',Pitch_init(posV),'
BlPitch(3)  - Blade 3 initial pitch (degrees) [unused for 2 blades]');
        A_text{index_fst(6)} = sprintf('%6.2f %s',Omr_init(posV),'
RotSpeed    - Initial or fixed rotor speed (rpm)');
        fid = fopen(FSTfile,'w');
        for i = 1:numel(A_text)
            if A_text{i+1} == -1
                fprintf(fid,'%s', A_text{i});
                break
            else
                fprintf(fid,'%s\n', A_text{i});
            end
        end
        fclose(fid);
        clear A_text i fid tline

        % Param_linearization.dat from Cases/linearization is modified with
the desired type of cotrol
        fid = fopen(DATfile,'r');
        i = 1;
        tline = fgetl(fid);
        A_text{i} = tline;
        while ischar(tline)
            i = i+1;
            tline = fgetl(fid);
            A_text{i} = tline;
        end
        fclose(fid);
        A_text{index_dat(1)} = sprintf('%7.0f %s',Type_control,'  TrimCase
- Trim case {1: find nacelle yaw, 2: find generator torque, 3: find collective
blade pitch} (switch) [used only when CalcStdy=True and GenDOF=True]');
        fid = fopen(DATfile,'w');
        for i = 1:numel(A_text)
            if A_text{i+1} == -1
                fprintf(fid,'%s', A_text{i});
                break
            else
                fprintf(fid,'%s\n', A_text{i});
            end
        end
        fclose(fid);
        clear A_text i fid tline

        %%%%% Linearization FAST model (sys_fast) %%%%%
        cd (FOLDER)
        !run.bat
        cd ..\..\..
        [LinModel] = GetSysFastSysturb(LINfile,Gbx,Wind_array(posV),NBlades);

        FAST.Azimuths = LinModel.Azimuths;
        FAST.Gbx = LinModel.Gbx;
        FAST.NBlades = LinModel.NBlades;
        FAST.NomSpeedArray(posV,1) = LinModel.NomSpeedArray;
```

```
        FAST.NomTorqueArray(posV,1) = LinModel.NomTorqueArray;
        FAST.PitchAngles(posV,1) = LinModel.PitchAngles;
        FAST.RotorSpeeds(1,posV) = LinModel.RotorSpeeds;
        FAST.SYSTURB.A(:,:,posV) = LinModel.SYSTURB.A;
        FAST.SYSTURB.B(:,:,posV) = LinModel.SYSTURB.B;
        FAST.SYSTURB.C(:,:,posV) = LinModel.SYSTURB.C;
        FAST.SYSTURB.D(:,:,posV) = LinModel.SYSTURB.D;
        FAST.SYSTURB.inputname = LinModel.SYSTURB.inputname;
        FAST.SYSTURB.outputname = LinModel.SYSTURB.outputname;
        FAST.SYSTURB.statename = LinModel.SYSTURB.statename;
        FAST.SteadyInput(:,posV) = LinModel.SteadyInput;
        FAST.SteadyOutput(:,posV) = LinModel.SteadyOutput;
        FAST.SteadyState(:,posV) = LinModel.SteadyState;
        FAST.Windspeeds(:,posV) = LinModel.Windspeeds;

        % Comprobamos que la linealización se ha hecho en la región correcta
        if (LinModel.NomTorqueArray > Tg_nom && Type_control==2)
            % Habíamos supuesto zona 1.5, 2 o 2.5 y deberíamos estar en zona 3

            Type_control=3;
            Pw_init(posV) = Pw_nom;                                % Wind Turbine
Power [W]
            Tg_init(posV) = Pw_init(posV)/(Omr_init(posV)*rpmrads*Gbx);
% Generator Torque [N·m]
            Lambda_init(posV) = Omr_init(posV)*rpmrads*R/Wind_array(posV);
            Cp_init(posV) =
2*Gbx*Tg_init(posV)*Omr_init(posV)*rpmrads/(rho*pi*R^2*Wind_array(posV)^3);
            cps_lambda0(:,posV) =
griddata(lambdas,betas,cps',Lambda_init(posV),betas);
            Pitch_init(posV) =
interp1(cps_lambda0(:,posV),betas,Cp_init(posV));          % Initial pitch
[deg]
            if isnan((Pitch_init(posV))) || Pitch_init(posV)<1e-5
                Pitch_init(posV)=beta_Z2;
            end
            Region(posV)=3;

            % linearization_gen.fst from Cases/linearization is modified with
the desired time and pitch
            fid = fopen(FSTfile,'r');
            i = 1;
            tline = fgetl(fid);
            A_text{i} = tline;
            while ischar(tline)
                i = i+1;
                tline = fgetl(fid);
                A_text{i} = tline;
            end
            fclose(fid);
            A_text{index_fst(3)} = sprintf('%6.1f %s',Pitch_init(posV),'
BlPitch(1)  - Blade 1 initial pitch (degrees)');
            A_text{index_fst(4)} = sprintf('%6.1f %s',Pitch_init(posV),'
BlPitch(2)  - Blade 2 initial pitch (degrees)');
            A_text{index_fst(5)} = sprintf('%6.1f %s',Pitch_init(posV),'
BlPitch(3)  - Blade 3 initial pitch (degrees) [unused for 2 blades]');
            A_text{index_fst(6)} = sprintf('%6.2f %s',Omr_init(posV),'
RotSpeed    - Initial or fixed rotor speed (rpm)');
            fid = fopen(FSTfile,'w');
            for i = 1:numel(A_text)
```

```
                if A_text{i+1} == -1
                    fprintf(fid,'%s', A_text{i});
                    break
                else
                    fprintf(fid,'%s\n', A_text{i});
                end
            end
            clear A_text i fid tline

            % Param_linearization.dat from Cases/linearization is modified
with the desired type of cotrol
            fid = fopen(DATfile,'r');
            i = 1;
            tline = fgetl(fid);
            A_text{i} = tline;
            while ischar(tline)
                i = i+1;
                tline = fgetl(fid);
                A_text{i} = tline;
            end
            fclose(fid);
            A_text{index_dat(1)} = sprintf('%7.0f %s',Type_control,'  TrimCase
- Trim case {1: find nacelle yaw, 2: find generator torque, 3: find collective
blade pitch} (switch) [used only when CalcStdy=True and GenDOF=True]');
            fid = fopen(DATfile,'w');
            for i = 1:numel(A_text)
                if A_text{i+1} == -1
                    fprintf(fid,'%s', A_text{i});
                    break
                else
                    fprintf(fid,'%s\n', A_text{i});
                end
            end
            clear A_text i fid tline

            % Linearization FAST model
            cd (FOLDER)
            !run.bat
            cd ..\..\..
            [LinModel] =
GetSysFastSysturb(LINfile,Gbx,Wind_array(posV),NBlades);

            FAST.Azimuths = LinModel.Azimuths;
            FAST.Gbx = LinModel.Gbx;
            FAST.NBlades = LinModel.NBlades;
            FAST.NomSpeedArray(posV,1) = LinModel.NomSpeedArray;
            FAST.NomTorqueArray(posV,1) = LinModel.NomTorqueArray;
            FAST.PitchAngles(posV,1) = LinModel.PitchAngles;
            FAST.RotorSpeeds(1,posV) = LinModel.RotorSpeeds;
            FAST.SYSTURB.A(:,:,posV) = LinModel.SYSTURB.A;
            FAST.SYSTURB.B(:,:,posV) = LinModel.SYSTURB.B;
            FAST.SYSTURB.C(:,:,posV) = LinModel.SYSTURB.C;
            FAST.SYSTURB.D(:,:,posV) = LinModel.SYSTURB.D;
            FAST.SYSTURB.inputname = LinModel.SYSTURB.inputname;
            FAST.SYSTURB.outputname = LinModel.SYSTURB.outputname;
            FAST.SYSTURB.statename = LinModel.SYSTURB.statename;
            FAST.SteadyInput(:,posV) = LinModel.SteadyInput;
            FAST.SteadyOutput(:,posV) = LinModel.SteadyOutput;
            FAST.SteadyState(:,posV) = LinModel.SteadyState;
```

```
            FAST.Windspeeds(:,posV) = LinModel.Windspeeds;

        end

        clear LinModel
end

clear known_beta known_Tg index_fst index_dat posV Type_control ans
```

Figuras

```
set(0,'defaultfigurecolor',[1 1 1])
cont15 = 1;
cont20 = 1;
cont25 = 1;
cont30 = 1;
POWER = FAST.SteadyOutput(strmatch('GenPwr',FAST.SYSTURB.outputname),:);
for posV = 1:length(FAST.Windspeeds)
    if Region(posV) == 1.5
        Wind15(cont15) = FAST.Windspeeds(posV);
        Speed15(cont15) = FAST.NomSpeedArray(posV);
        Torque15(cont15) = FAST.NomTorqueArray(posV);
        Pitch15(cont15) = FAST.PitchAngles(posV);
        Power15(cont15) = POWER(posV);
        Cp15(cont15) = Cp_init(posV);
        Region15(cont15) = Region(posV);
        cont15 = cont15 +1;
    elseif Region(posV) == 2
        Wind20(cont20) = FAST.Windspeeds(posV);
        Speed20(cont20) = FAST.NomSpeedArray(posV);
        Torque20(cont20) = FAST.NomTorqueArray(posV);
        Pitch20(cont20) = FAST.PitchAngles(posV);
        Power20(cont20) = POWER(posV);
        Cp20(cont20) = Cp_init(posV);
        Region20(cont20) = Region(posV);
        cont20 = cont20 +1;
    elseif Region(posV) == 2.5
        Wind25(cont25) = FAST.Windspeeds(posV);
        Speed25(cont25) = FAST.NomSpeedArray(posV);
        Torque25(cont25) = FAST.NomTorqueArray(posV);
        Pitch25(cont25) = FAST.PitchAngles(posV);
        Power25(cont25) = POWER(posV);
        Cp25(cont25) = Cp_init(posV);
        Region25(cont25) = Region(posV);
        cont25 = cont25 +1;
    else
        Wind30(cont30) = FAST.Windspeeds(posV);
        Speed30(cont30) = FAST.NomSpeedArray(posV);
        Torque30(cont30) = FAST.NomTorqueArray(posV);
        Pitch30(cont30) = FAST.PitchAngles(posV);
        Power30(cont30) = POWER(posV);
        Cp30(cont30) = Cp_init(posV);
        Region30(cont30) = Region(posV);
        cont30 = cont30 +1;
    end
end

figure(1); clf;
```

```
k(1)=subplot(3,1,1);
plot(FAST.Windspeeds,FAST.NomSpeedArray,'-b',...
    'LineWidth',0.65); hold on;
plot(Wind15,Speed15,'o',...
    'MarkerSize',5.5,...
    'MarkerFaceColor',[0.88627  0.95294  0.96863],...
    'MarkerEdgeColor',[0.88627  0.95294  0.96863]); hold on;
plot(Wind20,Speed20,'o',...
    'MarkerSize',5.5,...
    'MarkerFaceColor',[0.65098  0.87059  0.75686],...
    'MarkerEdgeColor',[0.65098  0.87059  0.75686]); hold on;
plot(Wind25,Speed25,'o',...
    'MarkerSize',5.5,...
    'MarkerFaceColor',[0.63922  0.84706  0.99216],...
    'MarkerEdgeColor',[0.63922  0.84706  0.99216]); hold on;
plot(Wind30,Speed30,'o',...
    'MarkerSize',5.5,...
    'MarkerFaceColor',[0.48235  0.59608  0.85098],...
    'MarkerEdgeColor',[0.48235  0.59608  0.85098]); hold on;
ylabel('\Omega_g (rad/s)');
ylim([min(FAST.NomSpeedArray)*0.95 max(FAST.NomSpeedArray)*1.05]);
k(2)=subplot(3,1,2);
plot(FAST.Windspeeds,FAST.NomTorqueArray,'-b',...
    'LineWidth',0.65); hold on;
plot(Wind15,Torque15,'o',...
    'MarkerSize',5.5,...
    'MarkerFaceColor',[0.88627  0.95294  0.96863],...
    'MarkerEdgeColor',[0.88627  0.95294  0.96863]); hold on;
plot(Wind20,Torque20,'o',...
    'MarkerSize',5.5,...
    'MarkerFaceColor',[0.65098  0.87059  0.75686],...
    'MarkerEdgeColor',[0.65098  0.87059  0.75686]); hold on;
plot(Wind25,Torque25,'o',...
    'MarkerSize',5.5,...
    'MarkerFaceColor',[0.63922  0.84706  0.99216],...
    'MarkerEdgeColor',[0.63922  0.84706  0.99216]); hold on;
plot(Wind30,Torque30,'o',...
    'MarkerSize',5.5,...
    'MarkerFaceColor',[0.48235  0.59608  0.85098],...
    'MarkerEdgeColor',[0.48235  0.59608  0.85098]); hold on;
ylabel('GenTorque (N·m)');
ylim([0 max(FAST.NomTorqueArray)*1.1]);
k(3)=subplot(3,1,3);
plot(FAST.Windspeeds,FAST.PitchAngles,'-b',...
    'LineWidth',0.65); hold on;
plot(Wind15,Pitch15,'o',...
    'MarkerSize',5.5,...
    'MarkerFaceColor',[0.88627  0.95294  0.96863],...
    'MarkerEdgeColor',[0.88627  0.95294  0.96863]); hold on;
plot(Wind20,Pitch20,'o',...
    'MarkerSize',5.5,...
    'MarkerFaceColor',[0.65098  0.87059  0.75686],...
    'MarkerEdgeColor',[0.65098  0.87059  0.75686]); hold on;
plot(Wind25,Pitch25,'o',...
    'MarkerSize',5.5,...
    'MarkerFaceColor',[0.63922  0.84706  0.99216],...
    'MarkerEdgeColor',[0.63922  0.84706  0.99216]); hold on;
plot(Wind30,Pitch30,'o',...
    'MarkerSize',5.5,...
```

```
    'MarkerFaceColor',[0.48235  0.59608  0.85098],...
    'MarkerEdgeColor',[0.48235  0.59608  0.85098]); hold on;
ylabel('\beta (rad)');
ylim([-0.05 max(FAST.PitchAngles)*1.05]);
linkaxes(k,'x');
xlim([FAST.Windspeeds(1)-1 FAST.Windspeeds(end)+1]);
xlabel('Wind speed (m/s)');

figure(2);clf;
k(1)=subplot(3,1,1);
plot(FAST.Windspeeds,POWER,'-b',...
    'LineWidth',0.65); hold on;
plot(Wind15,Power15,'o',...
    'MarkerSize',5.5,...
    'MarkerFaceColor',[0.88627  0.95294  0.96863],...
    'MarkerEdgeColor',[0.88627  0.95294  0.96863]); hold on;
plot(Wind20,Power20,'o',...
    'MarkerSize',5.5,...
    'MarkerFaceColor',[0.65098  0.87059  0.75686],...
    'MarkerEdgeColor',[0.65098  0.87059  0.75686]); hold on;
plot(Wind25,Power25,'o',...
    'MarkerSize',5.5,...
    'MarkerFaceColor',[0.63922  0.84706  0.99216],...
    'MarkerEdgeColor',[0.63922  0.84706  0.99216]); hold on;
plot(Wind30,Power30,'o',...
    'MarkerSize',5.5,...
    'MarkerFaceColor',[0.48235  0.59608  0.85098],...
    'MarkerEdgeColor',[0.48235  0.59608  0.85098]); hold on;
ylabel('Power (kW)');
ylim([0 max(POWER)*1.1]);
k(2)=subplot(3,1,2);
plot(FAST.Windspeeds,Cp_init,'-b',...
    'LineWidth',0.65); hold on;
plot(Wind15,Cp15,'o',...
    'MarkerSize',5.5,...
    'MarkerFaceColor',[0.88627  0.95294  0.96863],...
    'MarkerEdgeColor',[0.88627  0.95294  0.96863]); hold on;
plot(Wind20,Cp20,'o',...
    'MarkerSize',5.5,...
    'MarkerFaceColor',[0.65098  0.87059  0.75686],...
    'MarkerEdgeColor',[0.65098  0.87059  0.75686]); hold on;
plot(Wind25,Cp25,'o',...
    'MarkerSize',5.5,...
    'MarkerFaceColor',[0.63922  0.84706  0.99216],...
    'MarkerEdgeColor',[0.63922  0.84706  0.99216]); hold on;
plot(Wind30,Cp30,'o',...
    'MarkerSize',5.5,...
    'MarkerFaceColor',[0.48235  0.59608  0.85098],...
    'MarkerEdgeColor',[0.48235  0.59608  0.85098]); hold on;
ylabel('Cp');
ylim([0 max(Cp_init)*1.05]);
k(3)=subplot(3,1,3);
plot(FAST.Windspeeds,Region,'-b',...
    'LineWidth',0.65); hold on;
plot(Wind15,Region15,'o',...
    'MarkerSize',5.5,...
    'MarkerFaceColor',[0.88627  0.95294  0.96863],...
    'MarkerEdgeColor',[0.88627  0.95294  0.96863]); hold on;
plot(Wind20,Region20,'o',...
```

```
    'MarkerSize',5.5,...
    'MarkerFaceColor',[0.65098  0.87059  0.75686],...
    'MarkerEdgeColor',[0.65098  0.87059  0.75686]); hold on;
plot(Wind25,Region25,'o',...
    'MarkerSize',5.5,...
    'MarkerFaceColor',[0.63922  0.84706  0.99216],...
    'MarkerEdgeColor',[0.63922  0.84706  0.99216]); hold on;
plot(Wind30,Region30,'o',...
    'MarkerSize',5.5,...
    'MarkerFaceColor',[0.48235  0.59608  0.85098],...
    'MarkerEdgeColor',[0.48235  0.59608  0.85098]); hold on;
ylim([1 4]);
ylabel('Zonas de operación');
linkaxes(k,'x');
xlim([FAST.Windspeeds(1)-1 FAST.Windspeeds(end)+1]);
xlabel('Wind speed (m/s)');


figure(3); clf;
h(1)=subplot(3,1,1);
plot(FAST.NomSpeedArray,FAST.NomTorqueArray,'-b',...
    'LineWidth',0.65); hold on;
plot(Speed15,Torque15,'o',...
    'MarkerSize',5.5,...
    'MarkerFaceColor',[0.88627  0.95294  0.96863],...
    'MarkerEdgeColor',[0.88627  0.95294  0.96863]); hold on;
plot(Speed20,Torque20,'o',...
    'MarkerSize',5.5,...
    'MarkerFaceColor',[0.65098  0.87059  0.75686],...
    'MarkerEdgeColor',[0.65098  0.87059  0.75686]); hold on;
plot(Speed25,Torque25,'o',...
    'MarkerSize',5.5,...
    'MarkerFaceColor',[0.63922  0.84706  0.99216],...
    'MarkerEdgeColor',[0.63922  0.84706  0.99216]); hold on;
plot(Speed30,Torque30,'o',...
    'MarkerSize',5.5,...
    'MarkerFaceColor',[0.48235  0.59608  0.85098],...
    'MarkerEdgeColor',[0.48235  0.59608  0.85098]); hold on;
ylabel('GenTorque (N·m)');
ylim([0 max(FAST.NomTorqueArray)*1.1]);
h(2)=subplot(3,1,2);
plot(FAST.NomSpeedArray,POWER,'-b',...
    'LineWidth',0.65); hold on;
plot(Speed15,Power15,'o',...
    'MarkerSize',5.5,...
    'MarkerFaceColor',[0.88627  0.95294  0.96863],...
    'MarkerEdgeColor',[0.88627  0.95294  0.96863]); hold on;
plot(Speed20,Power20,'o',...
    'MarkerSize',5.5,...
    'MarkerFaceColor',[0.65098  0.87059  0.75686],...
    'MarkerEdgeColor',[0.65098  0.87059  0.75686]); hold on;
plot(Speed25,Power25,'o',...
    'MarkerSize',5.5,...
    'MarkerFaceColor',[0.63922  0.84706  0.99216],...
    'MarkerEdgeColor',[0.63922  0.84706  0.99216]); hold on;
plot(Speed30,Power30,'o',...
    'MarkerSize',5.5,...
    'MarkerFaceColor',[0.48235  0.59608  0.85098],...
    'MarkerEdgeColor',[0.48235  0.59608  0.85098]); hold on;
```

```
ylabel('Power (kW)');
ylim([0 max(POWER)*1.1]);
h(3)=subplot(3,1,3);
plot(FAST.NomSpeedArray,Region,'-b',...
    'LineWidth',0.65); hold on;
plot(Speed15,Region15,'o',...
    'MarkerSize',5.5,...
    'MarkerFaceColor',[0.88627  0.95294  0.96863],...
    'MarkerEdgeColor',[0.88627  0.95294  0.96863]); hold on;
plot(Speed20,Region20,'o',...
    'MarkerSize',5.5,...
    'MarkerFaceColor',[0.65098  0.87059  0.75686],...
    'MarkerEdgeColor',[0.65098  0.87059  0.75686]); hold on;
plot(Speed25,Region25,'o',...
    'MarkerSize',5.5,...
    'MarkerFaceColor',[0.63922  0.84706  0.99216],...
    'MarkerEdgeColor',[0.63922  0.84706  0.99216]); hold on;
plot(Speed30,Region30,'o',...
    'MarkerSize',5.5,...
    'MarkerFaceColor',[0.48235  0.59608  0.85098],...
    'MarkerEdgeColor',[0.48235  0.59608  0.85098]); hold on;
ylim([1 4]);
ylabel('Zonas de operación');
linkaxes(h,'x');
xlim([min(FAST.NomSpeedArray)*0.95 max(FAST.NomSpeedArray)*1.025]);
xlabel('\Omega_g (rad/s)');
```

Save the linear model as a structure

```
save('LinModel_20190820.mat','FAST');
clear all; clc;
load ('LinModel_20190820.mat')
Names = fieldnames(FAST);%List all variables under MyStructure
for nn = 1:length(Names)
  eval([Names{nn},' = FAST.',Names{nn},';']);% Assign data to original names
end
clear FAST Names nn
save('LinModel_20190820.mat')
```

ANEXO B: Estandarización LinModel

Modification of FAST for Inibode

Para poder usar los resultados de FAST con las herramientas de GAMESA, se modifican los nombres de las variables así como las unidades (sistema internacional)

```
load('LinModel_20190820.mat');
load('BLADEDnames.mat');

% Inputnames (different order). Cambio de unidades
AuxMatrix=diag(ones(1,size(SYSTURB.inputname,1)));
for i=1:size(SYSTURB.inputname,1)
    if i==1
        CB(i,:)=AuxMatrix(3,:); % m/s
    elseif i==2
        CB(i,:)=AuxMatrix(2,:); % rad
    else
        CB(i,:)=AuxMatrix(1,:); % N·m
    end
end
SYSTURB.inputname=names.input;
SteadyInput=CB*SteadyInput;
for i=1:length(Windspeeds)
    AuxB(:,:,i)=SYSTURB.B(:,:,i)*CB;
    AuxD(:,:,i)=SYSTURB.D(:,:,i)*CB;
end
SYSTURB.B=AuxB;
SYSTURB.D=AuxD;
clear AuxB AuxD i CB Aux AuxMatrix;

% Outputnames (same order). Cambio de unidades.
% Para nacelle x - position no tiene salida FAST por lo que no se cambia el
% valor pero EL NOMBRE DE LA VARIABLE (Nacelle fore-aft displacement) NO SE
% CORRESPONDE CON LO QUE REPRESENTA (TipSpdRat)
AuxMatrix=diag(ones(1,size(SYSTURB.outputname,1)));
for i=1:size(SYSTURB.outputname,1)
    if find(strfind(SYSTURB.outputname(i,:),'kN'))==1
        CB(i,:)=AuxMatrix(i,:)*1000;
    elseif find(strfind(SYSTURB.outputname(i,:),'kW'))==1
        CB(i,:)=AuxMatrix(i,:)*1000;
    elseif find(strfind(SYSTURB.outputname(i,:),'rpm'))==1
        CB(i,:)=AuxMatrix(i,:)*pi/30;
    elseif find(strfind(SYSTURB.outputname(i,:),'deg'))==1
        CB(i,:)=AuxMatrix(i,:)*pi/180;
    else
        CB(i,:)=AuxMatrix(i,:);
    end
end
SYSTURB.outputname=names.output;
SteadyOutput=CB*SteadyOutput;
for i=1:length(Windspeeds)
    AuxC(:,:,i)=CB*SYSTURB.C(:,:,i);
    AuxD(:,:,i)=CB*SYSTURB.D(:,:,i);
end
SYSTURB.C=AuxC;
SYSTURB.D=AuxD;
```

```
clear AuxC AuxD i CB Aux AuxMatrix;

clear names

save('LinModel_20190820_modified.mat')
```

ANEXO C: Función GetSysFastSysturb

```
function [LinModel] = GetSysFastSysturb(FASTLinName,Gbx,Wind,NBlades)

% Modified version of GetMats to be used as a function

% GetMats.m
% Written by J. Jonkman, NREL
% Last update: 10/06/2008 by B. Jonkman, NREL
% Compatible with FAST linear output (.lin) files generated using FAST v6.02a-
jmj.

% This m-file is used to read in the data written to A FAST linear output
%   (.lin) file, compute the state matrix, [A], at each of the equally-spaced
%   azimuth steps and their azimuth-average, along with their eigenvalues and
%   eigenvectors.

format short g;

ClearRootName = false;
% FASTLinName = [ RootName, '.lin'];   % FAST (.lin) filename


% ----------- Read in the matrices from the file, FASTLinName -------------

% Open the FAST linear file:
FileID = fopen( FASTLinName, 'r' );

% Abort if file is not found:
if ( FileID == -1 )
    disp( ['FAST linearization output file "', FASTLinName, '" not found.
Aborting.'] );
    clear ClearRootName RootName FASTLinName FileID;
    return;
end

% Tell the user what is running:
disp( ' ' );
disp( ['Running Eigenanalysis.m using "', FASTLinName, '"'] );
disp( 'Please wait...' );

% Read in (and ignore) the first portion of the file:
for Row = 1:10
    line   = fgetl( FileID );
end

% Read in the azimuth-average rotor speed, the steady state period of
solution,
%   then ignore the next three lines:
line = fgetl( FileID );
RotSpeed   = str2num( line(55:68) );    % in (rad/s)
line = fgetl( FileID );
if ( RotSpeed > 0 )
    Period = str2num( line(55:68) );    % in (sec)
else
    clear Period;
end
line = fgetl( FileID );
```

```
line = fgetl( FileID );
line = fgetl( FileID );

% Read in the number of equally spaced azimuth steps, the model order, the
%   number of active (enabled) DOFs, the number of control inputs, the
%   number of wind input disturbances, and the number of output measurements,
%   then ignore the next four lines:
line = fgetl( FileID );
NAzimStep  = str2num( line(55:58) );
line = fgetl( FileID );
MdlOrder   = str2num( line(55:58) );
line = fgetl( FileID );
NActvDOF   = str2num( line(55:58) );
line = fgetl( FileID );
NInputs    = str2num( line(55:58) );
line = fgetl( FileID );
NDisturbs  = str2num( line(55:58) );
line = fgetl( FileID );
NumOuts    = str2num( line(55:58) );
line = fgetl( FileID );
line = fgetl( FileID );
line = fgetl( FileID );
line = fgetl( FileID );

% Define the number of states:
N = 2*NActvDOF;                     % the number of states in the linearized
model {x}

% Initialize the matrices before reading them in:
DescStates        = cell (         NActvDOF ,1        );
Azimuth           = zeros(         NAzimStep,1        );
% %bjj start of change 10/2/08
Omega             = ones (         NAzimStep,1        )*RotSpeed;
OmegaDot          = zeros(         NAzimStep,1        );
%bjj end of change 10/2/08
xdop              = zeros(N                ,NAzimStep);
xop               = zeros(N                ,NAzimStep);
AMat              = zeros(N       ,N       ,NAzimStep);
if ( NInputs > 0 )
    DescCntrlInpt = cell (         NInputs  ,1        );
    BMat          = zeros(N       ,NInputs  ,NAzimStep);
else
    clear DescCntrlInpt BMat;
end
if ( NDisturbs > 0 )
    DescDisturbnc = cell (         NDisturbs,1        );
    BdMat         = zeros(N       ,NDisturbs,NAzimStep);
else
    clear DescDisturbnc BdMat;
end
if ( NumOuts > 0 )
    DescOutput    = cell (NumOuts           ,1        );
    OutName       = cell (NumOuts           ,1        );
    yop           = zeros(NumOuts           ,NAzimStep);
    CMat          = zeros(NumOuts ,N        ,NAzimStep);
    if ( NInputs > 0 )
        DMat      = zeros(NumOuts ,NInputs  ,NAzimStep);
    else
        clear DMat;
```

```
    end
    if ( NDisturbs > 0 )
        DdMat     = zeros(NumOuts ,NDisturbs,NAzimStep);
    else
        clear DdMat;
    end
else
    clear DescOutput OutName yop CMat DMat DdMat;
end
if ( MdlOrder == 2 )
    qd2op         = zeros(NActvDOF          ,NAzimStep);
    qdop          = zeros(NActvDOF          ,NAzimStep);
    qop           = zeros(NActvDOF          ,NAzimStep);
    MassMat       = zeros(NActvDOF,NActvDOF ,NAzimStep);
    DampMat       = zeros(NActvDOF,NActvDOF ,NAzimStep);
    StffMat       = zeros(NActvDOF,NActvDOF ,NAzimStep);
    if ( NInputs > 0 )
        FMat      = zeros(NActvDOF,NInputs  ,NAzimStep);
    else
        clear FMat;
    end
    if ( NDisturbs > 0 )
        FdMat     = zeros(NActvDOF,NDisturbs,NAzimStep);
    else
        clear FdMat;
    end
    if ( NumOuts > 0 )
        VelCMat   = zeros(NumOuts ,NActvDOF ,NAzimStep);
        DspCMat   = zeros(NumOuts ,NActvDOF ,NAzimStep);
    else
        clear VelCMat DspCMat;
    end
else
    clear qd2op qdop qop MassMat DampMat StffMat FMat FdMat VelCMat DspCMat;
end

% Read in the description of the model states, control inputs, input wind
%   disturbances, and output measurements:
if ( MdlOrder == 1 )
    for Row = 1:NActvDOF
        DescStates   {Row} = strtrim( fgetl( FileID ) );
    end
    line = fgetl( FileID );
else    % ( MdlOrder == 2 )
    for Row = 1:NActvDOF
        DescStates   {Row} = strtrim( fgetl( FileID ) );
    end
end
line = fgetl( FileID );
line = fgetl( FileID );
line = fgetl( FileID );
line = fgetl( FileID );
if ( NInputs > 0 )
    for Row = 1:NInputs
        DescCntrlInpt{Row} = strtrim( fgetl( FileID ) );
    end
else
    line = fgetl( FileID );
end
```

```
line = fgetl( FileID );
line = fgetl( FileID );
line = fgetl( FileID );
line = fgetl( FileID );
if ( NDisturbs > 0 )
    for Row = 1:NDisturbs
        DescDisturbnc{Row} = strtrim( fgetl( FileID ) );
    end
else
    line = fgetl( FileID );
end
line = fgetl( FileID );
line = fgetl( FileID );
line = fgetl( FileID );
line = fgetl( FileID );
if ( NumOuts > 0 )
    for Row = 1:NumOuts
        DescOutput   {Row} = strtrim( fgetl( FileID ) );
        EqlCol             = min(strfind(DescOutput{Row},'=')) + 1;
% find the index following the index of the first instance of string '='
        OutName      {Row} =
strtok(DescOutput{Row}(EqlCol:length(DescOutput{Row})));
    end
else
    line = fgetl( FileID );
end
line = fgetl( FileID );
line = fgetl( FileID );
line = fgetl( FileID );
line = fgetl( FileID );

% Loop through all azimuth steps:
for L = 1:NAzimStep

% Read in the azimuth step and reference azimuth angle from the azimuth title,
%   then ignore the two column headers:
    line = fgetl( FileID );
    Azimuth(L) = str2num( line(21:26) );   % in (deg)
    AzimB1Up   = str2num( line(60:65) );   % in (deg)
    line = fgetl( FileID );
    line = fgetl( FileID );

% Read in the vectors and matrices:
    if ( MdlOrder == 1 )

        for Row = 1:N
            line = fgetl( FileID );

            xdop(Row,L) = str2num( line( 1:10) );
            xop (Row,L) = str2num( line(14:23) );

            for Col = 1:N
                AMat         (Row,Col,L) = str2num( line(( 27     + 11*(
Col - 1 ) ):( 36     + 11*(               Col - 1 ) )) );
            end
            for Col = 1:NInputs
                BMat         (Row,Col,L) = str2num( line(( 27 + 2 + 11*(
N + Col - 1 ) ):( 36 + 2 + 11*(           N + Col - 1 ) )) );
            end
```

```
            if ( NInputs > 0 )      % control inputs exist
                for Col = 1:NDisturbs
                    BdMat    (Row,Col,L) = str2num( line(( 27 + 4 + 11*(
NInputs + N + Col - 1 ) ):( 36 + 4 + 11*( NInputs + N + Col - 1 ) )) );
                end
            else                    % no control inputs selected
                for Col = 1:NDisturbs
                    BdMat    (Row,Col,L) = str2num( line(( 27 + 2 + 11*(
N + Col - 1 ) ):( 36 + 2 + 11*(           N + Col - 1 ) )) );
                end
            end
        end

        if ( NumOuts > 0 )
            line = fgetl( FileID ); % ignore the header
            line = fgetl( FileID ); % for the output
            line = fgetl( FileID ); % matrices

            for Row = 1:NumOuts
                line = fgetl( FileID );

                yop (Row,L) = str2num( line( 1:10) );

                for Col = 1:N
                    CMat     (Row,Col,L) = str2num( line(( 27     + 11*(
Col - 1 ) ):( 36     + 11*(               Col - 1 ) )) );
                end
                for Col = 1:NInputs
                    DMat     (Row,Col,L) = str2num( line(( 27 + 2 + 11*(
N + Col - 1 ) ):( 36 + 2 + 11*(           N + Col - 1 ) )) );
                end
                if ( NInputs > 0 )      % control inputs exist
                    for Col = 1:NDisturbs
                        DdMat(Row,Col,L) = str2num( line(( 27 + 4 + 11*(
NInputs + N + Col - 1 ) ):( 36 + 4 + 11*( NInputs + N + Col - 1 ) )) );
                    end
                else                    % no control inputs selected
                    for Col = 1:NDisturbs
                        DdMat(Row,Col,L) = str2num( line(( 27 + 2 + 11*(
N + Col - 1 ) ):( 36 + 2 + 11*(           N + Col - 1 ) )) );
                    end
                end
            end
        end

    else    % ( MdlOrder == 2 )

        for Row = 1:NActvDOF
            line = fgetl( FileID );

            qd2op(Row,L) = str2num( line( 1:10) );
            qdop (Row,L) = str2num( line(14:23) );
            qop  (Row,L) = str2num( line(27:36) );

            for Col = 1:NActvDOF
                MassMat      (Row,Col,L) = str2num( line(( 40     + 11*(
Col - 1 ) ):( 49     + 11*(                        Col - 1 ) )) );
                DampMat      (Row,Col,L) = str2num( line(( 40 + 2 + 11*(
NActvDOF + Col - 1 ) ):( 49 + 2 + 11*(             NActvDOF + Col - 1 ) )) );
```

```
                StffMat      (Row,Col,L) = str2num( line(( 40 + 4 + 11*(
2*NActvDOF + Col - 1 ) ):( 49 + 4 + 11*(           2*NActvDOF + Col - 1 ) ))
);
            end
            for Col = 1:NInputs
                FMat         (Row,Col,L) = str2num( line(( 40 + 6 + 11*(
3*NActvDOF + Col - 1 ) ):( 49 + 6 + 11*(           3*NActvDOF + Col - 1 ) ))
);
            end
            if ( NInputs > 0 )      % control inputs exist
                for Col = 1:NDisturbs
                    FdMat    (Row,Col,L) = str2num( line(( 40 + 8 + 11*(
NInputs + 3*NActvDOF + Col - 1 ) ):( 49 + 8 + 11*( NInputs + 3*NActvDOF + Col
- 1 ) )) );
                end
            else                    % no control inputs selected
                for Col = 1:NDisturbs
                    FdMat    (Row,Col,L) = str2num( line(( 40 + 6 + 11*(
3*NActvDOF + Col - 1 ) ):( 49 + 6 + 11*(           3*NActvDOF + Col - 1 ) ))
);
                end
            end
        end

        if ( NumOuts > 0 )
            line = fgetl( FileID ); % ignore the header
            line = fgetl( FileID ); % for the output
            line = fgetl( FileID ); % matrices

            for Row = 1:NumOuts
                line = fgetl( FileID );

                yop (Row,L) = str2num( line( 1:10) );

                for Col = 1:NActvDOF
                    VelCMat  (Row,Col,L) = str2num( line(( 40 + 2 + 11*(
NActvDOF + Col - 1 ) ):( 49 + 2 + 11*(             NActvDOF + Col - 1 ) )) );
                    DspCMat  (Row,Col,L) = str2num( line(( 40 + 4 + 11*(
2*NActvDOF + Col - 1 ) ):( 49 + 4 + 11*(           2*NActvDOF + Col - 1 ) ))
);
                end
                for Col = 1:NInputs
                    DMat     (Row,Col,L) = str2num( line(( 40 + 6 + 11*(
3*NActvDOF + Col - 1 ) ):( 49 + 6 + 11*(           3*NActvDOF + Col - 1 ) ))
);
                end
                if ( NInputs > 0 )      % control inputs exist
                    for Col = 1:NDisturbs
                        DdMat(Row,Col,L) = str2num( line(( 40 + 8 + 11*(
NInputs + 3*NActvDOF + Col - 1 ) ):( 49 + 8 + 11*( NInputs + 3*NActvDOF + Col
- 1 ) )) );
                    end
                else                    % no control inputs selected
                    for Col = 1:NDisturbs
                        DdMat(Row,Col,L) = str2num( line(( 40 + 6 + 11*(
3*NActvDOF + Col - 1 ) ):( 49 + 6 + 11*(           3*NActvDOF + Col - 1 ) ))
);
                    end
                end
```

```
            end
        end

    end

% Ignore the blank lines:
    line = fgetl( FileID );
    line = fgetl( FileID );

end

% Close the FAST linear file:
fclose( FileID );


% ----------- Assemble a 1st order model from the 2nd order model ---------
% ----------- Azimuth average the matrices --------------------------------

if ( MdlOrder == 2 )

% Form the [A], [B], and [Bd] matrices (so that {xdot} = [A]{x} + [B]{u} +
%   [Bd]{ud}) at each azimuth step using the 2nd order state matrices:
% Also, form the 1st order [C] matrix using [VelC] and [DspC]:
    for L = 1:NAzimStep
        xdop     ( 1:N                                    , L ) = [ qdop(
1:NActvDOF, L ); qd2op( 1:NActvDOF, L ) ];
        xop      ( 1:N                                    , L ) = [ qop(
1:NActvDOF, L ); qdop(  1:NActvDOF, L ) ];
        AMat     ( 1:NActvDOF        , ( NActvDOF + 1 ):N, L ) =
eye(NActvDOF);
        AMat     ( ( NActvDOF + 1 ):N, 1:NActvDOF        , L ) = -
inv(MassMat(:,:,L))*StffMat(:,:,L);
        AMat     ( ( NActvDOF + 1 ):N, ( NActvDOF + 1 ):N, L ) = -
inv(MassMat(:,:,L))*DampMat(:,:,L);
        if ( NInputs > 0 )
            BMat ( ( NActvDOF + 1 ):N, 1:NInputs         , L ) =
inv(MassMat(:,:,L))*FMat   (:,:,L);
        end
        if ( NDisturbs > 0 )
            BdMat( ( NActvDOF + 1 ):N, 1:NDisturbs       , L ) =
inv(MassMat(:,:,L))*FdMat  (:,:,L);
        end
        if ( NumOuts > 0 )
            CMat ( 1:NumOuts         , 1:N               , L ) = [
DspCMat(1:NumOuts,1:NActvDOF,L),  VelCMat(1:NumOuts,1:NActvDOF,L) ];
        end
    end

% Find the azimuth-averaged linearized 2nd order state matrices:
    Avgqd2op       = zeros(NActvDOF,1        );   %
    Avgqdop        = zeros(NActvDOF,1        );   %
    Avgqop         = zeros(NActvDOF,1        );   %
    AvgMassMat     = zeros(NActvDOF,NActvDOF );   %
    AvgDampMat     = zeros(NActvDOF,NActvDOF );   %
    AvgStffMat     = zeros(NActvDOF,NActvDOF );   % first initialize to zero
    if ( NInputs > 0 )
        AvgFMat    = zeros(NActvDOF,NInputs  );   %
    else
        clear AvgFMat;
```

```
    end
    if ( NDisturbs > 0 )
        AvgFdMat   = zeros(NActvDOF,NDisturbs);   %
    else
        clear AvgFdMat;
    end
    if ( NumOuts > 0 )
        AvgVelCMat = zeros(NumOuts ,NActvDOF );   %
        AvgDspCMat = zeros(NumOuts ,NActvDOF );   %
    else
        clear AvgVelCMat AvgDspCMat;
    end
    for L = 1:NAzimStep
        Avgqd2op      (:  ) = Avgqd2op  (:  ) + qd2op  (:  ,L)/NAzimStep;
        Avgqdop       (:  ) = Avgqdop   (:  ) + qdop   (:  ,L)/NAzimStep;
        Avgqop        (:  ) = Avgqop    (:  ) + qop    (:  ,L)/NAzimStep;
        AvgMassMat    (:,:) = AvgMassMat(:,:) + MassMat(:,:,L)/NAzimStep;
        AvgDampMat    (:,:) = AvgDampMat(:,:) + DampMat(:,:,L)/NAzimStep;
        AvgStffMat    (:,:) = AvgStffMat(:,:) + StffMat(:,:,L)/NAzimStep;
        if ( NInputs > 0 )
            AvgFMat   (:,:) = AvgFMat   (:,:) + FMat   (:,:,L)/NAzimStep;
        end
        if ( NDisturbs > 0 )
            AvgFdMat  (:,:) = AvgFdMat  (:,:) + FdMat  (:,:,L)/NAzimStep;
        end
        if ( NumOuts > 0 )
            AvgVelCMat(:,:) = AvgVelCMat(:,:) + VelCMat(:,:,L)/NAzimStep;
            AvgDspCMat(:,:) = AvgDspCMat(:,:) + DspCMat(:,:,L)/NAzimStep;
        end
    end

else
    clear Avgqd2op Avgqdop Avgqop AvgMassMat AvgDampMat AvgStffMat AvgFMat
AvgFdMat AvgVelCMat AvgDspCMat;
end

% Find the azimuth-averaged linearized 1st order state matrices:
Avgxdop          = zeros(N       ,1         ); %
Avgxop           = zeros(N       ,1         ); %
AvgAMat          = zeros(N       ,N         ); %
if ( NInputs > 0 )
    AvgBMat      = zeros(N       ,NInputs   ); % first initialize to zero
else
    clear AvgBMat;
end
if ( NDisturbs > 0 )
    AvgBdMat     = zeros(N       ,NDisturbs ); %
else
    clear AvgBdMat;
end
if ( NumOuts > 0 )
    Avgyop       = zeros(NumOuts ,1         ); %
    AvgCMat      = zeros(NumOuts ,N         ); %
    if ( NInputs > 0 )
        AvgDMat  = zeros(NumOuts ,NInputs   ); %
    else
        clear AvgDMat;
    end
    if ( NDisturbs > 0 )
```

```
        AvgDdMat = zeros(NumOuts ,NDisturbs ); %
    else
        clear AvgDdMat;
    end
else
    clear Avgyop AvgCMat AvgDMat AvgDdMat;
end
for L = 1:NAzimStep
    Avgxdop         (:  ) = Avgxdop (:  ) + xdop (:  ,L)/NAzimStep;
    Avgxop          (:  ) = Avgxop  (:  ) + xop  (:  ,L)/NAzimStep;
    AvgAMat         (:,:) = AvgAMat (:,:) + AMat (:,:,L)/NAzimStep;
    if ( NInputs > 0 )
        AvgBMat     (:,:) = AvgBMat (:,:) + BMat (:,:,L)/NAzimStep;
    end
    if ( NDisturbs > 0 )
        AvgBdMat    (:,:) = AvgBdMat(:,:) + BdMat(:,:,L)/NAzimStep;
    end
    if ( NumOuts > 0 )
        Avgyop      (:  ) = Avgyop  (:  ) + yop  (:  ,L)/NAzimStep;
        AvgCMat     (:,:) = AvgCMat (:,:) + CMat (:,:,L)/NAzimStep;
        if ( NInputs > 0 )
            AvgDMat (:,:) = AvgDMat (:,:) + DMat (:,:,L)/NAzimStep;
        end
        if ( NDisturbs > 0 )
            AvgDdMat(:,:) = AvgDdMat(:,:) + DdMat(:,:,L)/NAzimStep;
        end
    end
end


% ----------- Find multi-blade coordinate (MBC) transformation indices ----

% Find the number of, and indices for, state triplets in the rotating
%   frame:
NRotTripletStates = 0;                  % first initialize to zero
if ( exist('RotTripletIndicesStates') )
    clear RotTripletIndicesStates;      % start from scratch if necessary
end
for I = 1:NActvDOF        % loop through all active (enabled) DOFs
    BldCol = strfind(DescStates{I},'blade');                    % find the
starting index of the string 'blade'
    if ( ~isempty(BldCol) )     % true if the DescStates{I} contains the
string 'blade'
        EqlCol = min(strfind(DescStates{I},'=')) + 1;           % find the
index following the index of the first instance of string '='
        [ K, OK ]     = str2num(DescStates{I}(BldCol+6));       % save the
blade number  for the initial   blade
        if ( OK && isreal(K) )  % true if the 2nd character after the string
'blade' is a real number (i.e., a blade number)
            Tmp              = zeros(1,3);                      % first
initialize to zero
            Tmp(        1,K) = I;                               % save the
index         for the initial   blade
            for J = (I+1):NActvDOF  % loop through all remaining active
(enabled) DOFs
                if ( ( length(DescStates{J}) >= BldCol ) && (
strcmpi(DescStates{J}(EqlCol:BldCol),DescStates{I}(EqlCol:BldCol)) ) )    %
true if we have the same state from a different blade
```

```
                    K = str2num(DescStates{J}(BldCol+6));       % save the
blade numbers for the remaining blades
                    Tmp(1,K) = J;                               % save the
indices       for the remaining blades
                end
            end                         % J - all remaining active (enabled) DOFs
            if ( all(Tmp) )     % true if all the elements of Tmp are nonzero;
thus, we found a triplet of rotating indices
                NRotTripletStates                               =
NRotTripletStates + 1;   % this  is  the number  of  state triplets in the
rotating frame
                RotTripletIndicesStates(NRotTripletStates,:) = Tmp;
% these are the indices for state triplets in the rotating frame
            end
        end
%bjj start of change 10/2/08
    else
        GeAzCol = strfind(DescStates{I},'DOF_GeAz');            % find the
starting index of the string 'DOF_GeAz'
        if ( ~isempty(GeAzCol) )     % true if the DescStates{I} contains the
string 'DOF_GeAz'
            Omega    (:) = xdop(I         ,:);
            OmegaDot(:) = xdop(I+NActvDOF,:);
        end
%bjj end of change 10/2/08
%bjj start of change 10/6/08
        DrTrCol = strfind(DescStates{I},'DOF_DrTr');            % find the
starting index of the string 'DOF_DrTr'
        if ( ~isempty(DrTrCol) )     % true if the DescStates{I} contains the
string 'DOF_DrTr'
            Omega    (:) = Omega    (:) + xdop(I         ,:)';  %This always
comes after DOF_GeAz so let's just add it here (it won't get written over
later).
            OmegaDot(:) = OmegaDot(:) + xdop(I+NActvDOF,:)';
        end
%bjj end of change 10/6/08
    end
end                 % I - all active (enabled) DOFs

% Find the number of, and indices for, control input triplets in the
%   rotating frame:
NRotTripletCntrlInpt = 0;               % first initialize to zero
if ( exist('RotTripletIndicesCntrlInpt') )
    clear RotTripletIndicesCntrlInpt;     % start from scratch if necessary
end
for I = 1:NInputs         % loop through all control inputs
    BldCol = strfind(DescCntrlInpt{I},'blade');                 % find the
starting index of the string 'blade'
    if ( ~isempty(BldCol) )     % true if the DescCntrlInpt{I} contains the
string 'blade'
        EqlCol = min(strfind(DescCntrlInpt{I},'=')) + 1;        % find the
index following the index of the first instance of string '='
        [ K, OK ]     = str2num(DescCntrlInpt{I}(BldCol+6));    % save the
blade number  for the initial   blade
        if ( OK && isreal(K) )  % true if the 2nd character after the string
'blade' is a real number (i.e., a blade number)
            Tmp              = zeros(1,3);                      % first
initialize to zero
```

```
            Tmp(        1,K) = I;                                   % save the
index         for the initial   blade
            for J = (I+1):NInputs % loop through all remaining control inputs
                if ( ( length(DescCntrlInpt{J}) >= BldCol ) && (
strcmpi(DescCntrlInpt{J}(EqlCol:BldCol),DescCntrlInpt{I}(EqlCol:BldCol)) ) )%
true if we have the same control input from a different blade
                    K = str2num(DescCntrlInpt{J}(BldCol+6));    % save the
blade numbers for the remaining blades
                    Tmp(1,K) = J;                               % save the
indices       for the remaining blades
                end
            end                     % J - all remaining active control inputs
            if ( all(Tmp) )     % true if all the elements of Tmp are nonzero;
thus, we found a triplet of rotating indices
                NRotTripletCntrlInpt                                =
NRotTripletCntrlInpt + 1;  % this  is  the number  of  control input triplets
in the rotating frame
                RotTripletIndicesCntrlInpt(NRotTripletCntrlInpt,:) = Tmp;
% these are the indices for control input triplets in the rotating frame
            end
        end
    end
end                 % I - all control inputs

% Find the number of, and indices for, output measurement triplets in the
%   rotating frame:
NRotTripletOutput = 0;                  % first initialize to zero
if ( exist('RotTripletIndicesOutput') )
    clear RotTripletIndicesOutput;      % start from scratch if necessary
end
for I = 1:NumOuts         % loop through all output measurements
    [ K, OK ]     = str2num(OutName{I}(length(OutName{I})));        % save the
blade number  for the initial   blade
    if ( OK && isreal(K) )  % true if the last character of string OutName{I}'
is a real number (i.e., a blade number)
        Tmp             = zeros(1,3);                               % first
initialize to zero
        Tmp(        1,K) = I;                                       % save the
index         for the initial   blade
        for J = (I+1):NumOuts     % loop through all remaining output
measurements
            if ( ( length(OutName{J}) == length(OutName{I}) ) && (
strcmpi(OutName{J}(1:length(OutName{J})-1),OutName{I}(1:length(OutName{I})-1))
) )      % true if we have the same output measurement from a different blade
                K = str2num(OutName{J}(length(OutName{J})));    % save the
blade numbers for the remaining blades
                Tmp(1,K) = J;                                   % save the
indices       for the remaining blades
            end
        end                     % J - all remaining active output measurements
        if ( all(Tmp) )     % true if all the elements of Tmp are nonzero;
thus, we found a triplet of rotating indices
            NRotTripletOutput                               = NRotTripletOutput +
1;      % this  is  the number  of  output measurement triplets in the
rotating frame
            RotTripletIndicesOutput(NRotTripletOutput,:) = Tmp;
% these are the indices for output measurement triplets in the rotating frame
        end
    end
```

```
end                 % I - all output measurements

% ----------- Clear some unneeded variables ------------------------------

% Clear some unneeded variables:
if ( ClearRootName )
    clear RootName;
end
%bjj start of change  10/2/08
%remove clear ClearRootName FASTLinName FileID line Row Col BldCol EqlCol I J
K L Tmp OK OutName Real1 Real2 Imag1 Imag2 ans;
%10/2/08 clear ClearRootName FASTLinName FileID line Row Col BldCol GeAzCol
EqlCol I J K L Tmp OK OutName Real1 Real2 Imag1 Imag2 ans;
clear ClearRootName FASTLinName FileID line Row Col BldCol GeAzCol DrTrCol
EqlCol I J K L Tmp OK OutName Real1 Real2 Imag1 Imag2 ans;
%bjj end of change 10/2/08

% ----------- We're finished ---------------------------------------------

% Tell the user that we are finished:
disp( '                                        ' );
disp( 'GetMats.m completed              ' );
disp( 'Type "who" to list available variables' );


% Create the linear system (Upna 2019)

% Las entradas y salidas obtenidas con GetMats se renombran según la
% nomenclatura que queremos emplear
for k = 1:NInputs
    in_names{k,1} = DescCntrlInpt{k}(12:end-18);
    in_steady(k,1) = str2num(DescCntrlInpt{k}(end-17:end-2));
end
for k = 1:NDisturbs
    in_names{k+NInputs,1} = DescDisturbnc{k}(12:end-32);
    in_steady(k+NInputs,1) = Wind;
end
for k = 1:length(DescOutput)
    out_names{k,1} = DescOutput{k}(11:end);
end
for k = 1:N
    if k>NActvDOF
        state_names{k,1} = sprintf('%s%s','First Derivative ',DescStates{k-
NActvDOF}(17:end));
    else
        state_names{k,1} = DescStates{k}(17:end);
    end
end

LinModel.Azimuths = Azimuth;
LinModel.Gbx = Gbx;
LinModel.NBlades = NBlades;
LinModel.NomSpeedArray = Omega*Gbx;
LinModel.NomTorqueArray =
in_steady(~cellfun(@isempty,strfind(in_names,'torque')));
LinModel.PitchAngles =
in_steady(~cellfun(@isempty,strfind(in_names,'pitch')));
LinModel.RotorSpeeds = Omega;
LinModel.SYSTURB.A = AMat;
```

```
LinModel.SYSTURB.B = [BMat BdMat];
LinModel.SYSTURB.C = CMat;
LinModel.SYSTURB.D = [DMat DdMat];
LinModel.SYSTURB.inputname = char(in_names);
LinModel.SYSTURB.outputname = char(out_names);
LinModel.SYSTURB.statename = char(state_names);
LinModel.SteadyInput = in_steady;
LinModel.SteadyOutput = yop;
LinModel.SteadyState = xop;
LinModel.Windspeeds = Wind;


end
```

ANEXO D: Validación Lazo Abierto

```
clear all;
clc;
close all;

cd D:\IDOIA\TFM\FAST
addpath('.\Cases\matlab_read');
```

Indicaciones de requisitos de los diferentes archivos para evitar errores

```
% Comprobar que el archivo .ipt tiene bien escrito la dirección del DETFILE
% y los .dat aerodinámicos (cylinder...)

% Comprobar que el archivo .fst empleado está en modo Run a time-marching
% simulation AnalMode = 1 (8).
% Comprobar que el archivo .fst tiene bien escrito la dirección del .dat de
% la torre (135), del .dat de las palas (157-159), del .itp (161)
% Ordenar las salidas del .fst según BLADED:
% "RotSpeed"                          - Rotor velocity (rpm)
% "GenSpeed"                          - Generator velocity (rpm)
% "GenTq"                             - Generator torque (kN·m)
% "BlPitch1"                          - Blade pitch angle (deg)
% "TipSpdRat"                         - Tip Speed Ratio
% "NcIMUTVxs"                         - Nacelle-x-velocity (m/s)
% "NcIMUTAxs"                                    - Nacelle-x-acceleration
(m^2/s)
% "RootMyb1"                          - Blade 1 flapwise moment My (kN·m)
% "RootMxb1"                          - Blade 1 edgewise moment Mx (kN·m)
% "LSShftMxa"                                    - LSS torque (EQUIVALENT TO THE
ROTOR TORQUE)
% "TwrBsMyt"                          - Tower base pitching torque (kN·m)
% "GenPwr"                            - Generator Power (kW)
```

Input data

```
load('LinModel_20190805.mat')

FOLDER='D:\IDOIA\TFM\FAST\Cases\control\simulink';
DETfile='D:\IDOIA\TFM\FAST\wind\deterministic\wind_det.wnd';
FSTfile='D:\IDOIA\TFM\FAST\Cases\control\simulink\FAST_input.fst';
IPTfile='D:\IDOIA\TFM\FAST\Cases\control\simulink\Wind.itp';
FSMfile='D:\IDOIA\TFM\FAST\Cases\control\simulink\FAST_input_SFunc.fsm';
OPTfile='D:\IDOIA\TFM\FAST\Cases\control\simulink\FAST_input_SFunc.opt';
OUTBfile='D:\IDOIA\TFM\FAST\Cases\control\simulink\FAST_input_SFunc.outb';
SIMUfile='D:\IDOIA\TFM\FAST\Cases\control\simulink\OpenLoop.slx';

Jr = 35444067;                      % Rotor inertia
load('power_curves_NREL5MW_v2.mat') % betas in deg
Omr_nom = 12.1;                     % rpm
Omr_min = 6.9;                      % rpm
Pw_nom = 5e6;                       % W
lambda_opt = 7.69;                  % -
beta_Z2=0;                          % beta [deg] in zone 2 and 2.5

% Wind determination
Wind_array=[13]; % Wind Speed vector wanted to validate [m/s]
```

```
% Disturbance definition
% Input disturbance: 'wind', 'pitch' or 'torque'
dist = 'wind';
% Type of disturbance: 'step', 'ramp', 'sin', 'turb', 'steps', 'lin'
type = 'step';
% Characteristics of the disturbance:
t_end=400;          % Total time simulated [s]
t_step = 200;       % Step time [s]. Only used when type = 'step'
t_p=0.02;           % Integration time step [s]
num_steps=2;        % Only used when type = 'steps'
freq=0.01;          % Only used when type = 'sin'
amplitude_array = 3;% 'wind' [m/s]; 'pitch' [rad], 'torque' [N·m]. It can be a
vector
```

Obtener información de los datos de entrada

```
sys_FAST=ss(SYSTURB.A,SYSTURB.B,SYSTURB.C,SYSTURB.D,'InputName',SYSTURB.inputn
ame,'OutputName',SYSTURB.outputname,'StateName',SYSTURB.statename);
missalignment=8;    % By default (wind determination)

% Read WTG parameters from FSTfile
parname{1}=sprintf('%s',' TipRad ');
parname{2}=sprintf('%s',' GenIner ');
parname{3}=sprintf('%s',' GBRatio ');
parname{4}=sprintf('%s',' TowerHt ');
parname{5}=sprintf('%s',' NumBl ');
parname{6}=sprintf('%s',' TMax ');
parname{7}=sprintf('%s',' DT ');
parname{8}=sprintf('%s',' BlPitch(1) ');
parname{9}=sprintf('%s',' BlPitch(2) ');
parname{10}=sprintf('%s',' BlPitch(3) ');
parname{11}=sprintf('%s',' RotSpeed ');
fst = textread(FSTfile,'%s','delimiter','\n');
cont=0;
parindex=zeros(1,11);
parline=cell(1,5);
par=zeros(1,5);
index_fst=zeros(1,6);
for j=1:length(parname)
    parindex(j)=find(~cellfun(@isempty,strfind(fst,parname{j})));
    if j<6
        parline{j}=cell2mat(fst(parindex(j)));
        par(j)=eval(parline{j}(1:strfind(parline{j},parname{j})-1));
    else
        cont=cont+1;
        index_fst(cont)=parindex(j);
    end
end
R = par(1);             % Wind turbine radio [m]
Jg = par(2);            % Generator inertia
Gbx = par(3);           % Gearbox ratio
Jtot = Jr + Gbx^2*Jg;   % Total inertia
htower = par(4);        % Tower height [m]
NBlades = par(5);       % Number of blades
clear fst parindex parline par j parname

% Read conditions from OPTfile
```

```
parname{1}=sprintf('%s','Air density');
opt = textread(OPTfile,'%s','delimiter','\n');
parindex=zeros(1,1);
parline=cell(1,1);
par=zeros(1,1);
for j=1:length(parname)
    parindex(j)=find(~cellfun(@isempty,strfind(opt,parname{j})));
    parline{j}=cell2mat(opt(parindex(j)));
    par(j)=str2double(strtok(parline(j)));
end
rho=par(1);              % Air density [kg/m^3]
clear aer parindex parline par j parname

% Units conversion
radsrpm=30/pi;           % From rad/s to rpm
rpmrads=1/radsrpm;       % From rpm a rad/s
degrad=pi/180;           % From degrees to rad
raddeg=1/degrad;         % From rad a grados (deg)
% Units used in FAST: [rad], [rpm], [m/s], [m/s^2], [N], [N·m], [W], [s]

Tg_nom = Pw_nom/(Omr_nom*rpmrads*Gbx); % N·m
cp_max = griddata(lambdas,betas,cps',lambda_opt,beta_Z2);
K_opt = 1/(2*Gbx)*rho*pi*R^2*cp_max*(R^3/lambda_opt^3)*rpmrads^2/Gbx^2;

s = tf('s');
ncolor=2*length(Wind_array)*length(amplitude_array);
colors = distinguishable_colors(ncolor);
TMax=t_end;
```

OPEN LOOP VALIDATION

```
set(groot,'defaultFigureColor','w')

cont=0;
numfig_init=1;

Windindex =
find(~cellfun(@isempty,strfind(cellstr(SYSTURB.inputname),'wind')));
Torqueindex =
find(~cellfun(@isempty,strfind(cellstr(SYSTURB.inputname),'torque')));
Pitchindex =
find(~cellfun(@isempty,strfind(cellstr(SYSTURB.inputname),'pitch')));

for posV=1:length(Wind_array)

    LINindex = find(Windspeeds== Wind_array(posV));
    col_ext=colors(posV,:);

    % FAST_input.fst from Cases/control/simulink is modified
    fid = fopen(FSTfile,'r');
    i = 1;
    tline = fgetl(fid);
    A_text{i} = tline;
    while ischar(tline)
        i = i+1;
        tline = fgetl(fid);
        A_text{i} = tline;
    end
```

```
    fclose(fid);
    A_text{index_fst(1)} = sprintf('%6.0f %s',t_end,'     TMax        - Total
run time (s)');
    A_text{index_fst(2)} = sprintf('%8.4f %s',t_p,'   DT          -
Integration time step (s)');
    A_text{index_fst(3)} = sprintf('%6.2f %s',PitchAngles(LINindex)*raddeg,'
BlPitch(1)  - Blade 1 initial pitch (degrees)');
    A_text{index_fst(4)} = sprintf('%6.2f %s',PitchAngles(LINindex)*raddeg,'
BlPitch(2)  - Blade 2 initial pitch (degrees)');
    A_text{index_fst(5)} = sprintf('%6.2f %s',PitchAngles(LINindex)*raddeg,'
BlPitch(3)  - Blade 3 initial pitch (degrees) [unused for 2 blades]');
    A_text{index_fst(6)} = sprintf('%6.2f %s',RotorSpeeds(LINindex)*radsrpm,'
RotSpeed    - Initial or fixed rotor speed (rpm)');
    fid = fopen(FSTfile,'w');
    for i = 1:numel(A_text)
        if A_text{i+1} == -1
            fprintf(fid,'%s', A_text{i});
            break
        else
            fprintf(fid,'%s\n', A_text{i});
        end
    end
    clear A_text

    for posAmp=1:length(amplitude_array)
        amplitude = amplitude_array(posAmp);
        cont=cont+1;
        numfig=numfig_init;
        col_int1=colors(2*cont-1,:);
        col_int2=colors(2*cont,:);

        % Legend text for the different figures
        legendTextDist{cont}=sprintf('%s %s%s%2.0f%s%3.0f',dist,type,'
V=',Wind_array(posV),'; Amp=',amplitude);
        legendText{2*cont-1}=sprintf('%s%2.0f%s%3.0f','Simulado
V=',Wind_array(posV),'; Amp=',amplitude);
        legendText{2*cont}=sprintf('%s%2.0f%s%3.0f','LinModel
V=',Wind_array(posV),'; Amp=',amplitude);

        % LIN_FAST
        t_lin = (0:t_p:t_end)';
        if strcmp('step',type)==1
            u_lin(1:t_step/t_p) = 0;
            u_lin(t_step/t_p+1:t_end/t_p+1) = amplitude;
        elseif strcmp('ramp',type)==1
            for k=1:length(t_lin)
                u_lin(k) = amplitude*k*t_p/t_end;
            end
        elseif strcmp('sin',type)==1
            for k=1:length(t_lin)
                u_lin(k) = amplitude*sin(2*pi*freq*t_p*k);
            end
        elseif strcmp('turb',type)==1
            for k=1:length(t_lin)
                u_lin(k) = amplitude*rand(1);
            end
        elseif strcmp('steps',type)==1
            for k=1:length(t_lin)
```

```
                u_lin(k) =
amplitude*floor(k*t_p/(t_end/num_steps))/(num_steps-1);
            end
        end
        if strcmp('wind',dist)==1
            wind_det(t_end, t_step, t_p, Wind_array(posV),
Wind_array(posV)+amplitude, num_steps, freq, missalignment, type, DETfile);
            y_lin = lsim(sys_FAST(:,Windindex,LINindex),u_lin,t_lin);
            Tg_init(1:t_end/t_p+1) = NomTorqueArray(LINindex);
            Pitch_init(1:t_end/t_p+1) = PitchAngles(LINindex);
        elseif strcmp('pitch',dist)==1
            wind_det(t_end, 0, t_p, Wind_array(posV), Wind_array(posV),
num_steps, 0, missalignment, 'lin', DETfile);
            y_lin = lsim(sys_FAST(:,Pitchindex,LINindex),u_lin,t_lin);
            Tg_init(1:t_end/t_p+1) = NomTorqueArray(LINindex);
            Pitch_init(1:t_end/t_p+1) = u_lin(:) + PitchAngles(LINindex);
        elseif strcmp('torque',dist)==1
            wind_det(t_end, 0, t_p, Wind_array(posV), Wind_array(posV),
num_steps, 0, missalignment, 'lin', DETfile);
            y_lin = lsim(sys_FAST(:,Torqueindex,LINindex),u_lin,t_lin);
            Tg_init(1:t_end/t_p+1) = u_lin(:) + NomTorqueArray(LINindex);
            Pitch_init(1:t_end/t_p+1) = PitchAngles(LINindex);
        end
        lin_fast = zeros(size(y_lin,1),size(SYSTURB.outputname,1));
        for k=1:size(SYSTURB.outputname,1)
            lin_fast(:,k) = y_lin(:,k) + SteadyOutput(k,LINindex);
        end

        % SIM_FAST (SIMULINK)
        cd (FOLDER)
        input_fast = FSTfile; % archivo a ejecutar
        Read_FAST_Input
        sim('OpenLoop')
        cd ..\..\..
        FASTBinaryFile = OUTBfile;
        ReadFASTBinaryAndMAP;
        t_sim = Channels(:,ChannelNamesIndex('Time')); % s
        sim_fast = zeros(size(t_sim,1),size(SYSTURB.outputname,1));
        for k=1:size(SYSTURB.outputname,1)
            sim_fast(:,k) =
Channels(:,ChannelNamesIndex(strtok(SYSTURB.outputname(k,:))));
        end
        clear DOF_BE DOF_BF DOF_DrTr DOF_GeAz DOF_Hv DOF_P DOF_R DOF_RFrl
DOF_Sg DOF_Sw DOF_Teet DOF_TFA1 DOF_TFA2 DOF_TFrl DOF_TSS1 DOF_TSS2 DOF_Y
DOF_Yaw;
        clear ChannelIndex ChannelNames ChannelNamesChannelUnits
ChannelNamesIndex ChannelUnits
        clear DT k i Initialized NDF tline UnitConversionFactorToSI Units

        % Open Loop results representation
        figure(numfig);
        h(1)=subplot(3,1,1); hold all;
        if strcmp('wind',dist)==1
            plot(t_lin,u_lin+Windspeeds(LINindex),'color',col_int1); hold on;
            legend(legendTextDist);
            ylabel('V (m/s)');
            title('Wind Speed (m/s)');
        elseif strcmp('pitch',dist)==1
            plot(t_lin,u_lin+PitchAngles(LINindex),'color',col_int1); hold on;
```

```
            legend(legendTextDist);
            ylabel('\beta (rad)');
            title('Pitch Angle (rad)');
        elseif strcmp('torque',dist)==1
            plot(t_lin,u_lin+NomTorqueArray(LINindex),'color',col_int1); hold
on;
            legend(legendTextDist);
            ylabel('T_g (N·m)');
            title('Torque (N·m)');
        end
        h(2)=subplot(3,1,[2,3]); hold all;

plot(t_sim,sim_fast(:,strncmp('GenSpeed',cellstr(SYSTURB.outputname),8)),'colo
r',col_int1); hold on;

plot(t_lin,lin_fast(:,strncmp('GenSpeed',cellstr(SYSTURB.outputname),8)),':','
color',col_int2); hold on;
        legend(legendText);
        ylabel('\Omega_g (rpm)');
        linkaxes(h,'x');
        xlabel('Time (s)');
        numfig=numfig+1;

        figure(numfig);
        h(1)=subplot(2,2,[1 2]);  hold all;
        if strcmp('wind',dist)==1
            plot(t_lin,u_lin+Windspeeds(LINindex),'color',col_int1); hold on;
            legend(legendTextDist);
            ylabel('V (m/s)');
            title('Wind Speed (m/s)');
        elseif strcmp('pitch',dist)==1
            plot(t_lin,u_lin+PitchAngles(LINindex),'color',col_int1); hold on;
            legend(legendTextDist);
            ylabel('\beta (rad)');
            title('Pitch Angle (rad)');
        elseif strcmp('torque',dist)==1
            plot(t_lin,u_lin+NomTorqueArray(LINindex),'color',col_int1); hold
on;
            legend(legendTextDist);
            ylabel('T_g (N·m)');
            title('Torque (N·m)');
        end
        h(2)=subplot(2,2,3); hold all;

plot(t_sim,sim_fast(:,strncmp('GenTq',cellstr(SYSTURB.outputname),5)),'color',
col_int1); hold on;

plot(t_lin,lin_fast(:,strncmp('GenTq',cellstr(SYSTURB.outputname),5)),':','col
or',col_int2); hold on;
        legend(legendText);
        ylabel('T_g (kN·m)');
        xlabel('Time (s)');
        h(3)=subplot(2,2,4); hold all;

plot(t_sim,sim_fast(:,strncmp('BlPitch1',cellstr(SYSTURB.outputname),8)),'colo
r',col_int1); hold on;

plot(t_lin,lin_fast(:,strncmp('BlPitch1',cellstr(SYSTURB.outputname),8)),':','
color',col_int2); hold on;
```

```
        legend(legendText);
        ylabel('\beta (deg)');
        linkaxes(h,'x');
        xlabel('Time (s)');
        numfig=numfig+1;

        figure(numfig);
        h(1)=subplot(3,1,1); hold all;
        if strcmp('wind',dist)==1
            plot(t_lin,u_lin+Windspeeds(LINindex),'color',col_int1); hold on;
            legend(legendTextDist);
            ylabel('V (m/s)');
            title('Wind Speed (m/s)');
        elseif strcmp('pitch',dist)==1
            plot(t_lin,u_lin+PitchAngles(LINindex),'color',col_int1); hold on;
            legend(legendTextDist);
            ylabel('\beta (rad)');
            title('Pitch Angle (rad)');
        elseif strcmp('torque',dist)==1
            plot(t_lin,u_lin+NomTorqueArray(LINindex),'color',col_int1); hold
on;
            legend(legendTextDist);
            ylabel('T_g (N·m)');
            title('Torque (N·m)');
        end
        h(2)=subplot(3,1,[2,3]); hold all;

plot(t_sim,sim_fast(:,strncmp('GenPwr',cellstr(SYSTURB.outputname),6)),'color'
,col_int1); hold on;

plot(t_lin,lin_fast(:,strncmp('GenPwr',cellstr(SYSTURB.outputname),6)),':','co
lor',col_int2); hold on;
        legend(legendText);
        ylabel('Power (kW)');
        linkaxes(h,'x');
        xlabel('Time (s)');
    end
end
```

ANEXO E: Validación Diagrama de Bode

```
% Permite validar la linealización del sistema LinModel para un vector de
vientos (Wind_array) a través del diagrama de Bode
addpath('.\Cases\matlab_read');
```

Input data

```
load('LinModel_20190531.mat')

FOLDER='D:\IDOIA\TFM\FAST\Cases\control\simulink';
DETfile='D:\IDOIA\TFM\FAST\wind\deterministic\wind_det.wnd';
FSTfile='D:\IDOIA\TFM\FAST\Cases\control\simulink\FAST_input.fst';
IPTfile='D:\IDOIA\TFM\FAST\Cases\control\simulink\Wind.itp';
FSMfile='D:\IDOIA\TFM\FAST\Cases\control\simulink\FAST_input_SFunc.fsm';
OPTfile='D:\IDOIA\TFM\FAST\Cases\control\simulink\FAST_input_SFunc.opt';
OUTBfile='D:\IDOIA\TFM\FAST\Cases\control\simulink\FAST_input_SFunc.outb';
SIMUfile='D:\IDOIA\TFM\FAST\Cases\control\simulink\OpenLoop.slx';

Jr = 35444067;                      % Rotor inertia
load('power_curves_NREL5MW_v2.mat') % betas in deg
Omr_nom = 12.1;                     % rpm
Omr_min = 6.9;                      % rpm
Pw_nom = 5e6;                       % W
lambda_opt = 7.69;                  % -
beta_Z2=0;                          % beta [deg] in zone 2 and 2.5

% Wind determination
Wind_array=3:25;  % Wind Speed vector wanted to validate [m/s]

% Disturbance definition
% Input disturbance: 'wind', 'pitch' or 'torque'
dist = 'wind';
% Type of disturbance: 'step', 'ramp', 'sin', 'turb', 'steps', 'lin'
type = 'sin';
% Characteristics of the disturbance:
t_end=600;          % Total time simulated [s]
t_step = 200;       % Step time [s]. Only used when type = 'step'
t_p=0.02;           % Integration time step [s]
num_steps=2;        % Only used when type = 'steps'
amplitude = 1;      % 'wind' [m/s]; 'pitch' [rad], 'torque' [N·m]

% We want to validate the Bode diagram of the LinModel, therefore various sin
disturbance inputs at different frequencies must be introduced
Nmuestras_array=logspace(log10(5),log10(50000),100);
for j=1:length(Nmuestras_array)
    freq_array(j)=1/(Nmuestras_array(j)*t_p);
end
```

Obtaining information from the input data

```
sys_FAST=ss(SYSTURB.A,SYSTURB.B,SYSTURB.C,SYSTURB.D,'InputName',SYSTURB.inputn
ame,'OutputName',SYSTURB.outputname,'StateName',SYSTURB.statename);
missalignment=8;    % By default (wind determination)

% Read WTG parameters from FSTfile
parname{1}=sprintf('%s',' TipRad ');
```

```
parname{2}=sprintf('%s',' GenIner ');
parname{3}=sprintf('%s',' GBRatio ');
parname{4}=sprintf('%s',' TowerHt ');
parname{5}=sprintf('%s',' NumBl ');
parname{6}=sprintf('%s',' TMax ');
parname{7}=sprintf('%s',' DT ');
parname{8}=sprintf('%s',' BlPitch(1) ');
parname{9}=sprintf('%s',' BlPitch(2) ');
parname{10}=sprintf('%s',' BlPitch(3) ');
parname{11}=sprintf('%s',' RotSpeed ');
fst = textread(FSTfile,'%s','delimiter','\n');
cont=0;
parindex=zeros(1,11);
parline=strings(1,5);
par=zeros(1,5);
index_fst=zeros(1,6);
for j=1:length(parname)
    parindex(j)=find(~cellfun(@isempty,strfind(fst,parname{j})));
    if j<6
        parline{j}=cell2mat(fst(parindex(j)));
        par(j)=eval(parline{j}(1:strfind(parline{j},parname{j})-1));
    else
        cont=cont+1;
        index_fst(cont)=parindex(j);
    end
end
R = par(1);             % Wind turbine radio [m]
Jg = par(2);            % Generator inertia
Gbx = par(3);           % Gearbox ratio
Jtot = Jr + Gbx^2*Jg;   % Total inertia
htower = par(4);        % Tower height [m]
NBlades = par(5);       % Number of blades
clear fst parindex parline par j parname

% Read conditions from OPTfile
parname{1}=sprintf('%s','Air density');
opt = textread(OPTfile,'%s','delimiter','\n');
parindex=zeros(1,1);
parline=strings(1,1);
par=zeros(1,1);
for j=1:length(parname)
    parindex(j)=find(~cellfun(@isempty,strfind(opt,parname{j})));
    parline{j}=cell2mat(opt(parindex(j)));
    par(j)=str2double(strtok(parline(j)));
end
rho=par(1);             % Air density [kg/m^3]
clear aer parindex parline par j parname

% Units conversion
radsrpm=30/pi;          % From rad/s to rpm
rpmrads=1/radsrpm;      % From rpm a rad/s
degrad=pi/180;          % From degrees to rad
raddeg=1/degrad;        % From rad a grados (deg)
% Units used in FAST: [rad], [rpm], [m/s], [m/s^2], [N], [N·m], [W], [s]

Tg_nom = Pw_nom/(Omr_nom*rpmrads*Gbx); % N·m
cp_max = griddata(lambdas,betas,cps',lambda_opt,beta_Z2);
K_opt = 1/(2*Gbx)*rho*pi*R^2*cp_max*(R^3/lambda_opt^3)*rpmrads^2/Gbx^2;
```

```
s = tf('s');
ncolor=2*length(Wind_array)*length(freq_array);
colors = distinguishable_colors(ncolor);
TMax=t_end;
```

OPEN LOOP VALIDATION – Bode Diagram

```
set(groot,'defaultFigureColor','w')

cont=0;
numfig_int=1;

Windindex = find(contains(cellstr(SYSTURB.inputname),'wind'));
Torqueindex = find(contains(cellstr(SYSTURB.inputname),'torque'));
Pitchindex = find(contains(cellstr(SYSTURB.inputname),'pitch'));

Dist_max = zeros(length(freq_array),length(Wind_array));
Dist_I_max = zeros(length(freq_array),length(Wind_array));
Dist_min = zeros(length(freq_array),length(Wind_array));
Dist_I_min = zeros(length(freq_array),length(Wind_array));
Dist_amplitude = zeros(length(freq_array),length(Wind_array));
Omr_max = zeros(length(freq_array),length(Wind_array));
Omr_I_max = zeros(length(freq_array),length(Wind_array));
Omr_min = zeros(length(freq_array),length(Wind_array));
Omr_I_min = zeros(length(freq_array),length(Wind_array));
Omr_amplitude = zeros(length(freq_array),length(Wind_array));
delta_fase_Omr = zeros(length(freq_array),length(Wind_array));
Omg_max = zeros(length(freq_array),length(Wind_array));
Omg_I_max = zeros(length(freq_array),length(Wind_array));
Omg_min = zeros(length(freq_array),length(Wind_array));
Omg_I_min = zeros(length(freq_array),length(Wind_array));
Omg_amplitude = zeros(length(freq_array),length(Wind_array));
delta_fase_Omg = zeros(length(freq_array),length(Wind_array));
Tg_max = zeros(length(freq_array),length(Wind_array));
Tg_I_max = zeros(length(freq_array),length(Wind_array));
Tg_min = zeros(length(freq_array),length(Wind_array));
Tg_I_min = zeros(length(freq_array),length(Wind_array));
Tg_amplitude = zeros(length(freq_array),length(Wind_array));
delta_fase_Tg = zeros(length(freq_array),length(Wind_array));
Beta_max = zeros(length(freq_array),length(Wind_array));
Beta_I_max = zeros(length(freq_array),length(Wind_array));
Beta_min = zeros(length(freq_array),length(Wind_array));
Beta_I_min = zeros(length(freq_array),length(Wind_array));
Beta_amplitude = zeros(length(freq_array),length(Wind_array));
delta_fase_Beta = zeros(length(freq_array),length(Wind_array));
Pw_max = zeros(length(freq_array),length(Wind_array));
Pw_I_max = zeros(length(freq_array),length(Wind_array));
Pw_min = zeros(length(freq_array),length(Wind_array));
Pw_I_min = zeros(length(freq_array),length(Wind_array));
Pw_amplitude = zeros(length(freq_array),length(Wind_array));
delta_fase_Pw = zeros(length(freq_array),length(Wind_array));
Gain_Omr = zeros(length(freq_array),length(Wind_array));
Gain_Omg = zeros(length(freq_array),length(Wind_array));
Gain_Tg = zeros(length(freq_array),length(Wind_array));
Gain_Beta = zeros(length(freq_array),length(Wind_array));
Gain_Pw = zeros(length(freq_array),length(Wind_array));

for posV=1:length(Wind_array)
```

```
    LINindex = find(WindSpeeds== Wind_array(posV));

    % FAST_input.fst from Cases/control/simulink is modified
    fid = fopen(FSTfile,'r');
    i = 1;
    tline = fgetl(fid);
    A_text{i} = tline;
    while ischar(tline)
        i = i+1;
        tline = fgetl(fid);
        A_text{i} = tline;
    end
    fclose(fid);
    A_text{index_fst(1)} = sprintf('%6.0f %s',t_end,'     TMax        - Total
run time (s)');
    A_text{index_fst(2)} = sprintf('%8.4f %s',t_p,'   DT          -
Integration time step (s)');
    A_text{index_fst(3)} = sprintf('%6.2f %s',PitchAngles(LINindex)*raddeg,'
BlPitch(1)  - Blade 1 initial pitch (degrees)');
    A_text{index_fst(4)} = sprintf('%6.2f %s',PitchAngles(LINindex)*raddeg,'
BlPitch(2)  - Blade 2 initial pitch (degrees)');
    A_text{index_fst(5)} = sprintf('%6.2f %s',PitchAngles(LINindex)*raddeg,'
BlPitch(3)  - Blade 3 initial pitch (degrees) [unused for 2 blades]');
    A_text{index_fst(6)} = sprintf('%6.2f %s',RotorSpeeds(LINindex)*radsrpm,'
RotSpeed    - Initial or fixed rotor speed (rpm)');
    fid = fopen(FSTfile,'w');
    for i = 1:numel(A_text)
        if A_text{i+1} == -1
            fprintf(fid,'%s', A_text{i});
            break
        else
            fprintf(fid,'%s\n', A_text{i});
        end
    end
    clear A_text

    for posF=1:length(freq_array)
        freq = freq_array(posF);
        Nmuestras=round(Nmuestras_array(posF));

        cont=cont+1;
        numfig=numfig_int;
        col_int1=colors(2*cont-1,:);
        col_int2=colors(2*cont,:);

        % Legend text for the different figures
        legendTextDist{cont}=sprintf('%s %s%s%2.0f%s%3.4f',dist,type,'
V=',Wind_array(posV),'; F=',freq);
        legendText{4*cont-3}=sprintf('%s%2.0f%s%3.4f','SIMULATED
V=',Wind_array(posV),'; F=',freq);
        legendText{4*cont-2}=sprintf('%s%2.0f%s%3.4f','LinModel
V=',Wind_array(posV),'; F=',freq);
        legendText{4*cont-1}=sprintf('%s%2.0f%s%3.4f','MAX
V=',Wind_array(posV),'; F=',freq);
        legendText{4*cont}=sprintf('%s%2.0f%s%3.4f','MIN
V=',Wind_array(posV),'; F=',freq);

        % LIN_FAST
```

```
        t_lin = (0:t_p:t_end)';
        if strcmp('step',type)==1
            u_lin(1:t_step/t_p) = 0;
            u_lin(t_step/t_p+1:t_end/t_p+1) = amplitude;
        elseif strcmp('ramp',type)==1
            for k=1:length(t_lin)
                u_lin(k) = amplitude*k*t_p/t_end;
            end
        elseif strcmp('sin',type)==1
            for k=1:length(t_lin)
                u_lin(k) = amplitude*sin(2*pi*freq*t_p*k);
            end
        elseif strcmp('turb',type)==1
            for k=1:length(t_lin)
                u_lin(k) = amplitude*rand(1);
            end
        elseif strcmp('steps',type)==1
            for k=1:length(t_lin)
                u_lin(k) =
amplitude*floor(k*t_p/(t_end/num_steps))/(num_steps-1);
            end
        end
        if strcmp('wind',dist)==1
            wind_det(t_end, t_step, t_p, Wind_array(posV),
Wind_array(posV)+amplitude, num_steps, freq, missalignment, type, DETfile);
            y_lin = lsim(sys_FAST(:,Windindex,LINindex),u_lin,t_lin);
            Tg_init(1:t_end/t_p+1) = NomTorqueArray(LINindex);
            Pitch_init(1:t_end/t_p+1) = PitchAngles(LINindex);
        elseif strcmp('pitch',dist)==1
            wind_det(t_end, 0, t_p, Wind_array(posV), Wind_array(posV),
num_steps, 0, missalignment, 'lin', DETfile);
            y_lin = lsim(sys_FAST(:,Pitchindex,LINindex),u_lin,t_lin);
            Tg_init(1:t_end/t_p+1) = NomTorqueArray(LINindex);
            Pitch_init(1:t_end/t_p+1) = u_lin(:) + PitchAngles(LINindex);
        elseif strcmp('torque',dist)==1
            wind_det(t_end, 0, t_p, Wind_array(posV), Wind_array(posV),
num_steps, 0, missalignment, 'lin', DETfile);
            y_lin = lsim(sys_FAST(:,Torqueindex,LINindex),u_lin,t_lin);
            Tg_init(1:t_end/t_p+1) = u_lin(:) + NomTorqueArray(LINindex);
            Pitch_init(1:t_end/t_p+1) = PitchAngles(LINindex);
        end
        lin_fast = zeros(size(y_lin,1),size(SYSTURB.outputname,1));
        for k=1:size(SYSTURB.outputname,1)
            lin_fast(:,k) = y_lin(:,k) + SteadyOutput(k,LINindex);
        end

        % SIM_FAST (SIMULINK)
        cd (FOLDER)
        input_fast = FSTfile; % archivo a ejecutar
        Read_FAST_Input
        sim('OpenLoop')
        cd ..\..\..
        t_sim = Time;
        sim_fast = zeros(size(t_sim,1),size(SYSTURB.outputname,1));
        for k=1:size(SYSTURB.outputname,1)
            sim_fast(:,k) = OutData(:,k);
        end
```

```
        clear DOF_BE DOF_BF DOF_DrTr DOF_GeAz DOF_Hv DOF_P DOF_R DOF_RFrl
DOF_Sg DOF_Sw DOF_Teet DOF_TFA1 DOF_TFA2 DOF_TFrl DOF_TSS1 DOF_TSS2 DOF_Y
DOF_Yaw;
        clear ChannelIndex ChannelNames ChannelNamesChannelUnits
ChannelNamesIndex ChannelUnits
        clear DT k i Initialized NDF tline UnitConversionFactorToSI Units

        % Disturbance values
        if strcmp('wind',dist)==1
            Dist_sim = u_lin+WindSpeeds(LINindex);
        elseif strcmp('pitch',dist)==1
            Dist_sim = u_lin+PitchAngles(LINindex);
        elseif strcmp('torque',dist)==1
            Dist_sim = u_lin+NomTorqueArray(LINindex);
        end
        [Dist_max(posF,posV), Dist_I_max(posF,posV)]= max(Dist_sim(end-
Nmuestras:end));
        [Dist_min(posF,posV), Dist_I_min(posF,posV)]= min(Dist_sim(end-
Nmuestras:end));
        Dist_amplitude(posF,posV)= Dist_max(posF,posV)-Dist_min(posF,posV);

        % Rotor speed values
        Omr_sim =
sim_fast(:,strncmp('RotSpeed',cellstr(SYSTURB.outputname),8));
        [Omr_max(posF,posV), Omr_I_max(posF,posV)]= max(Omr_sim(end-
Nmuestras:end));
        [Omr_min(posF,posV), Omr_I_min(posF,posV)]= min(Omr_sim(end-
Nmuestras:end));
        Omr_amplitude(posF,posV)= Omr_max(posF,posV)-Omr_min(posF,posV);
        delta_fase_Omr(posF,posV)=(t_sim(end-Nmuestras-
1+Dist_I_max(posF,posV))-t_sim(end-Nmuestras-
1+Omr_I_max(posF,posV)))*freq*360;

        % Gen speed values
        Omg_sim =
sim_fast(:,strncmp('GenSpeed',cellstr(SYSTURB.outputname),8));
        [Omg_max(posF,posV), Omg_I_max(posF,posV)]= max(Omg_sim(end-
Nmuestras:end));
        [Omg_min(posF,posV), Omg_I_min(posF,posV)]= min(Omg_sim(end-
Nmuestras:end));
        Omg_amplitude(posF,posV)= Omg_max(posF,posV)-Omg_min(posF,posV);
        delta_fase_Omg(posF,posV)= (t_sim(end-Nmuestras-
1+Dist_I_max(posF,posV))-t_sim(end-Nmuestras-
1+Omg_I_max(posF,posV)))*freq*360;

        % Gen Torque values
        Tg_sim = sim_fast(:,strncmp('GenTq',cellstr(SYSTURB.outputname),5));
        [Tg_max(posF,posV), Tg_I_max(posF,posV)]= max(Tg_sim(end-
Nmuestras:end));
        [Tg_min(posF,posV), Tg_I_min(posF,posV)]= min(Tg_sim(end-
Nmuestras:end));
        Tg_amplitude(posF,posV)= Tg_max(posF,posV)-Tg_min(posF,posV);
        delta_fase_Tg(posF,posV)=(t_sim(end-Nmuestras-
1+Dist_I_max(posF,posV))-t_sim(end-Nmuestras-1+Tg_I_max(posF,posV)))*freq*360;

        % Pitch angle values
        Beta_sim =
sim_fast(:,strncmp('BlPitch1',cellstr(SYSTURB.outputname),8));
```

```
        [Beta_max(posF,posV), Beta_I_max(posF,posV)]= max(Beta_sim(end-
Nmuestras:end));
        [Beta_min(posF,posV), Beta_I_min(posF,posV)]= min(Beta_sim(end-
Nmuestras:end));
        Beta_amplitude(posF,posV)= Beta_max(posF,posV)-Beta_min(posF,posV);
        delta_fase_Beta(posF,posV)=(t_sim(end-Nmuestras-
1+Dist_I_max(posF,posV))-t_sim(end-Nmuestras-
1+Beta_I_max(posF,posV)))*freq*360;
        % Power values
        Pw_sim = sim_fast(:,strncmp('GenPwr',cellstr(SYSTURB.outputname),6));
        [Pw_max(posF,posV), Pw_I_max(posF,posV)]= max(Pw_sim(end-
Nmuestras:end));
        [Pw_min(posF,posV), Pw_I_min(posF,posV)]= min(Pw_sim(end-
Nmuestras:end));
        Pw_amplitude(posF,posV)= Pw_max(posF,posV)-Pw_min(posF,posV);
        delta_fase_Pw(posF,posV)=(t_sim(end-Nmuestras-
1+Dist_I_max(posF,posV))-t_sim(end-Nmuestras-1+Pw_I_max(posF,posV)))*freq*360;


Gain_Omr(posF,posV)=20*log10(Omr_amplitude(posF,posV)/Dist_amplitude(posF,posV
));

Gain_Omg(posF,posV)=20*log10(Omg_amplitude(posF,posV)/Dist_amplitude(posF,posV
));

Gain_Tg(posF,posV)=20*log10(Tg_amplitude(posF,posV)/Dist_amplitude(posF,posV))
;

Gain_Beta(posF,posV)=20*log10(Beta_amplitude(posF,posV)/Dist_amplitude(posF,po
sV));

Gain_Pw(posF,posV)=20*log10(Pw_amplitude(posF,posV)/Dist_amplitude(posF,posV))
;

    end
end
save('data_23_08_2019_WIND.mat')
clear all;
clc; close all;
load('data_23_08_2019_WIND.mat');

for k=1:length(Wind_array)
    for j=1:length(freq_array)
        delta_fase_Omr(j,k)=delta_fase_Omr(j,k)-
360*floor((delta_fase_Omr(j,k)+180)/360);
        delta_fase_Omg(j,k)=delta_fase_Omg(j,k)-
360*floor((delta_fase_Omg(j,k)+180)/360);
        delta_fase_Tg(j,k)=delta_fase_Tg(j,k)-
360*floor((delta_fase_Tg(j,k)+180)/360);
        delta_fase_Beta(j,k)=delta_fase_Beta(j,k)-
360*floor((delta_fase_Beta(j,k)+180)/360);
        delta_fase_Pw(j,k)=delta_fase_Pw(j,k)-
360*floor((delta_fase_Pw(j,k)+180)/360);
    end
end
clear j k
```

FIGURE REPRESENTATION

```
w = freq_array*(2*pi);

ncolor = 2*length(freq_array);

colors = distinguishable_colors(ncolor);

if strcmp('wind',dist)==1
    Inputindex = Windindex;
    Inputname = sprintf('%s','Wind (m/s)');
elseif strcmp('pitch',dist)==1
    Inputindex = Pitchindex;
    Inputname = sprintf('%s','Pitch (rad)');
elseif strcmp('torque',dist)==1
    Inputindex = Torqueindex;
    Inputname = sprintf('%s','Torque (N·m)');
end

Wind_figures = [4 9 13 19];

for cont = 1:length(Wind_figures)
    posV=find(Windspeeds==Wind_figures(cont));
    posB=find(Wind_array==Wind_figures(cont));
    col1=colors(2*cont-1,:);
    col2=colors(2*cont,:);
    Legend{2*cont-1}=sprintf('%s%2.0f%s','LinModel
V=',Windspeeds(posV),'m/s');
    Legend{2*cont}=sprintf('%s%2.0f%s','Simulated V=',Windspeeds(posV),'m/s');

    figure(1); hold all;
[mag,phase]=Bode(sys_FAST(strncmp('RotSpeed',cellstr(SYSTURB.outputname),8),In
putindex,posV),w); hold on;
    for j=1:length(phase)
        phase(:,:,j)=phase(:,:,j)-360*floor((phase(:,:,j)+180)/360);
    end
    h(1)=subplot(2,1,1);
    semilogx(freq_array, 20*log10(squeeze(mag)),':','Color',col1); hold on;
    semilogx(freq_array, Gain_Omr(:,posB),'-*','Color',col2,'MarkerSize',3);
hold on;
    xlabel('Hz');
    ylabel('dB');
    legend(Legend);
    TitleText = sprintf('%s%s%s','Bode ',Inputname,' - \Omega_r (rpm)');
    title(TitleText);
    h(2)=subplot(2,1,2);
    semilogx(freq_array, squeeze(phase),':','Color',col1); hold on;
    semilogx(freq_array, delta_fase_Omr(:,posB),'-
*','Color',col2,'MarkerSize',3);
    xlabel('Hz');
    ylabel('°');
    linkaxes(h,'x');
    xlim([min(freq_array) max(freq_array)])
    clear mag phase j

    figure(2); hold all;
[mag,phase]=Bode(sys_FAST(strncmp('GenSpeed',cellstr(SYSTURB.outputname),8),In
putindex,posV),w);
    for j=1:length(phase)
        phase(:,:,j)=phase(:,:,j)-360*floor((phase(:,:,j)+180)/360);
```

```
    end
    h(1)=subplot(2,1,1);
    semilogx(freq_array, 20*log10(squeeze(mag)),':','Color',col1); hold on;
    semilogx(freq_array, Gain_Omg(:,posB),'-*','Color',col2,'MarkerSize',3);
hold on;
    xlabel('Hz');
    ylabel('dB');
    legend(Legend);
    TitleText = sprintf('%s%s%s','Bode ',Inputname,' - \Omega_g (rpm)');
    title(TitleText);
    h(2)=subplot(2,1,2);
    semilogx(freq_array, squeeze(phase),':','Color',col1); hold on;
    semilogx(freq_array, delta_fase_Omg(:,posB),'-
*','Color',col2,'MarkerSize',3);
    xlabel('Hz');
    ylabel('°');
    linkaxes(h,'x');
    xlim([min(freq_array) max(freq_array)])
    clear mag phase j

    figure(3); hold all;
[mag,phase]=Bode(sys_FAST(strncmp('GenPwr',cellstr(SYSTURB.outputname),6),Inpu
tindex,posV),w);
    for j=1:length(phase)
        phase(:,:,j)=phase(:,:,j)-360*floor((phase(:,:,j)+180)/360);
    end
    h(1)=subplot(2,1,1);
    semilogx(freq_array, 20*log10(squeeze(mag)),':','Color',col1); hold on;
    semilogx(freq_array, Gain_Pw(:,posB),'-*','Color',col2,'MarkerSize',3);
hold on;
    xlabel('Hz');
    ylabel('dB');
    legend(Legend);
    TitleText = sprintf('%s%s%s','Bode ',Inputname,' - Power (kW)');
    title(TitleText);
    h(2)=subplot(2,1,2);
    semilogx(freq_array, squeeze(phase),':','Color',col1); hold on;
    semilogx(freq_array, delta_fase_Pw(:,posB),'-
*','Color',col2,'MarkerSize',3);
    xlabel('Hz');
    ylabel('°');
    linkaxes(h,'x');
    xlim([min(freq_array) max(freq_array)])
    clear mag phase j

end
```

ANEXO F: Validación Lazo Cerrado

```
clear all
clc
close all

cd D:\IDOIA\TFM\FAST
addpath('.\Cases\matlab_read');
```

Input data

```
LinModel='LinModel_20190805.mat';
load('ClosedLoop_And_Controllers.mat');
% load ('Controllers.mat');

FOLDER='D:\IDOIA\TFM\FAST\Cases\control\simulink';
DETfile='D:\IDOIA\TFM\FAST\wind\deterministic\wind_det.wnd';
FSTfile='D:\IDOIA\TFM\FAST\Cases\control\simulink\FAST_input.fst';
IPTfile='D:\IDOIA\TFM\FAST\Cases\control\simulink\Wind.itp';
FSMfile='D:\IDOIA\TFM\FAST\Cases\control\simulink\FAST_input_SFunc.fsm';
OPTfile='D:\IDOIA\TFM\FAST\Cases\control\simulink\FAST_input_SFunc.opt';
OUTBfile='D:\IDOIA\TFM\FAST\Cases\control\simulink\FAST_input_SFunc.outb';
SIMUfile='D:\IDOIA\TFM\FAST\Cases\control\simulink\OpenLoop.slx';

Jr = 35444067;                      % Rotor inertia
load('power_curves_NREL5MW_v2.mat') % betas in deg
Omr_nom = 12.1;                     % rpm
Omr_min = 6.9;                      % rpm
Pw_nom = 5e6;                       % W
lambda_opt = 7.69;                  % -
beta_Z2=0;                          % beta [deg] in zone 2 and 2.5
```

Obtaining information from the input data

```
load(LinModel)

missalignment=8;    % By default (wind determination)

% Read WTG parameters from FSTfile
parname{1}=sprintf('%s',' TipRad ');
parname{2}=sprintf('%s',' GenIner ');
parname{3}=sprintf('%s',' GBRatio ');
parname{4}=sprintf('%s',' TowerHt ');
parname{5}=sprintf('%s',' NumBl ');
parname{6}=sprintf('%s',' TMax ');
parname{7}=sprintf('%s',' DT ');
parname{8}=sprintf('%s',' BlPitch(1) ');
parname{9}=sprintf('%s',' BlPitch(2) ');
parname{10}=sprintf('%s',' BlPitch(3) ');
parname{11}=sprintf('%s',' RotSpeed ');
fst = textread(FSTfile,'%s','delimiter','\n');
cont=0;
parindex=zeros(1,11);
parline=strings(1,5);
par=zeros(1,5);
index_fst=zeros(1,6);
for j=1:length(parname)
```

```
    parindex(j)=find(~cellfun(@isempty,strfind(fst,parname{j})));
    if j<6
        parline{j}=cell2mat(fst(parindex(j)));
        par(j)=eval(parline{j}(1:strfind(parline{j},parname{j})-1));
    else
        cont=cont+1;
        index_fst(cont)=parindex(j);
    end
end
R = par(1);             % Wind turbine radio [m]
Jg = par(2);            % Generator inertia
Gbx = par(3);           % Gearbox ratio
Jtot = Jr + Gbx^2*Jg;   % Total inertia
htower = par(4);        % Tower height [m]
NBlades = par(5);       % Number of blades
clear fst parindex parline par j parname

% Read conditions from OPTfile
parname{1}=sprintf('%s','Air density');
opt = textread(OPTfile,'%s','delimiter','\n');
parindex=zeros(1,1);
parline=strings(1,1);
par=zeros(1,1);
for j=1:length(parname)
    parindex(j)=find(~cellfun(@isempty,strfind(opt,parname{j})));
    parline{j}=cell2mat(opt(parindex(j)));
    par(j)=str2double(strtok(parline(j)));
end
rho=par(1);             % Air density [kg/m^3]
clear aer parindex parline par j parname

% Units conversion
radsrpm=30/pi;          % From rad/s to rpm
rpmrads=1/radsrpm;      % From rpm a rad/s
degrad=pi/180;          % From degrees to rad
raddeg=1/degrad;        % From rad a grados (deg)
% Units used in FAST: [rad], [rpm], [m/s], [m/s^2], [N], [N·m], [W], [s]

Omg_nom = Omr_nom*Gbx;
Omg_min= Omr_min*Gbx;
Tg_nom = Pw_nom/(Omr_nom*rpmrads*Gbx); % N·m
cp_max = griddata(lambdas,betas,cps',lambda_opt,beta_Z2);
K_opt = 1/(2*Gbx)*rho*pi*R^2*cp_max*(R^3/lambda_opt^3)*rpmrads^2/Gbx^2;
Tg_minspeed = K_opt*(Omg_min)^2; % N·m
Tg_nomspeed = K_opt*(Omg_nom)^2; % N·m

w = logspace(-3,1,2000)*(2*pi);
bode_opt = bodeoptions;
bode_opt.FreqUnits = 'Hz';
bode_opt.MagUnits = 'dB';
bode_opt.PhaseUnits = 'deg';
bode_opt.PhaseWrapping = 'off';
bode_opt.PhaseMatching = 'on';
bode_opt.PhaseMatchingFreq = 0.1;
bode_opt.PhaseMatchingValue = 0;
set(groot,'defaultFigureColor','w');

% Zonas de operación
Kopt_reg = [0, 1, 0, 0];
```

```
Z3_reg = [0, 0, 0, 1];
Z3_reg_op = [1, 1, 1, 0];
Z15_reg_op = [0, 1, 1, 1];
Z25_reg_op = [1, 1, 0, 1];

% Filtros
T_Omg = 2*pi*10;
T_Omr = 2*pi*0.1;
T_Acc = 2*pi*0.1;
T_Tg = 2*pi*1;
T_B = 2*pi*1;

sys_FAST = WTG.CL;
```

Simulation Simulink (fast model)

```
close all
% Wind_array = [4 7 9 12 16 22];
% Wind_array = [13 16 19 22];
% Wind_array = [4 6 8 10];
Wind_array = 25;
% Input disturbance: 'wind', 'pitch' or 'torque'
dist = 'wind';
% Type of disturbance: 'step', 'ramp', 'sin', 'turb', 'steps', 'lin',
'ramp_delay'
type = 'ramp_delay';
% Simulation parameters
t_end = 1000;           % Total time simulated [s]
t_p = 0.02;           % Integration time step [s]
t_step = 50;         % Step time [s]. Only used when type = 'step' or
'ramp_delay'
% Simulink disturbances and inputs:
dWg = 0;
dNacx = 0;
% Characteristics of the disturbance:
amplitude = -22;       % 'wind' [m/s]; 'pitch' [rad], 'torque' [N·m]. It can
be a vector
num_steps = 4;        % Only used when type = 'steps'
freq = 0.01;          % Only used when type = 'sin'

numfig_init = 1;
colors =  distinguishable_colors(2*length(Wind_array));

for cont = 1:length(Wind_array)

    V_init = Wind_array (cont);
    posV_init = find(Windspeeds==V_init);
    legendTextDist{cont}=sprintf('%s %s%s%2.0f%s%3.0f',dist,type,'
V=',V_init,'; Amp=',amplitude);
    legendText{2*cont-1}=sprintf('%s%2.0f%s%3.0f','Simulated V=',V_init,';
Amp=',amplitude);
    legendText{2*cont}=sprintf('%s%2.0f%s%3.0f','LinModel V=',V_init,';
Amp=',amplitude);
    if strcmp(dist,'wind')
        V_end = V_init + amplitude;
    else
        V_end = V_init;
    end
```

```
    col_int1 = colors(2*cont-1,:);
    col_int2 = colors(2*cont,:);

    % Initial conditions
    Omg0 = V_init*lambda_opt/R*Gbx*radsrpm;
    if Omg0>Omg_nom             % Starting in Z3
        Omg0=Omg_nom;
        region_init=3;
        deltabeta_init=0.42;
        Tg_init = Tg_nom;
    elseif Omg0<Omg_min
        Omg0=Omg_min;
        region_init=1.5;
        deltabeta_init=0;
        Tg_init = 0;
    else
        Omg0=Omg_min;
        region_init=2;
        deltabeta_init=0;
        Tg_init = Tg_minspeed;
    end

    % Modify FAST .fst file
    fid = fopen(FSTfile,'r');
    i = 1;
    tline = fgetl(fid);
    A_text{i} = tline;
    while ischar(tline)
        i = i+1;
        tline = fgetl(fid);
        A_text{i} = tline;
    end
    fclose(fid);
    A_text{index_fst(1)} = sprintf('%6.0f %s',t_end,'     TMax        - Total
run time (s)');
    A_text{index_fst(2)} = sprintf('%8.4f %s',t_p,'   DT          -
Integration time step (s)');
    A_text{index_fst(3)} = sprintf('%6.2f %s',PitchAngles(posV_init)*raddeg,'
BlPitch(1)  - Blade 1 initial pitch (degrees)');
    A_text{index_fst(4)} = sprintf('%6.2f %s',PitchAngles(posV_init)*raddeg,'
BlPitch(2)  - Blade 2 initial pitch (degrees)');
    A_text{index_fst(5)} = sprintf('%6.2f %s',PitchAngles(posV_init)*raddeg,'
BlPitch(3)  - Blade 3 initial pitch (degrees) [unused for 2 blades]');
    A_text{index_fst(6)} = sprintf('%6.2f %s',RotorSpeeds(posV_init)*radsrpm,'
RotSpeed    - Initial or fixed rotor speed (rpm)');
    fid = fopen(FSTfile,'w');
    for i = 1:numel(A_text)
        if A_text{i+1} == -1
            fprintf(fid,'%s', A_text{i});
            break
        else
            fprintf(fid,'%s\n', A_text{i});
        end
    end
    clear A_text

    % LIN_FAST
    t_lin = (0:t_p:t_end)';
    if strcmp('step',type)==1
```

```
        u_lin(1:t_step/t_p) = 0;
        u_lin(t_step/t_p+1:t_end/t_p+1) = amplitude;
        GenSpeed_reference(1:t_step/t_p) = SteadyOutput(2,posV_init);
        GenSpeed_reference(t_step/t_p+1:t_end/t_p+1) = SteadyOutput(2,V_end-
2);
    elseif strcmp('ramp',type)==1
        for k=1:length(t_lin)
            u_lin(k) = amplitude*k*t_p/t_end;
            if SteadyOutput(2,V_init-2)<800
                GenSpeed_reference(k) = Omg_min;
            else
                GenSpeed_reference(k) = Omg_nom;
            end
        end
    elseif strcmp('ramp_delay',type)==1
        for k=1:length(t_lin)
            if k*t_p<t_step
                u_lin(k) = 0;
                if SteadyOutput(2,posV_init+floor(amplitude*k*t_p/t_end))<800
                    GenSpeed_reference(k) = Omg_min;
                else
                    GenSpeed_reference(k) = Omg_nom;
                end
            else
                u_lin(k) = amplitude*(k*t_p-t_step)/(t_end-t_step);
                if SteadyOutput(2,posV_init+round(amplitude*k*t_p/t_end))<800
                    GenSpeed_reference(k) = Omg_min;
                else
                    GenSpeed_reference(k) = Omg_nom;
                end
            end
        end
    elseif strcmp('sin',type)==1
        for k=1:length(t_lin)
            u_lin(k) = amplitude*sin(2*pi*freq*t_p*k);
            GenSpeed_reference(k) = SteadyOutput(2,V_init-2);
        end
    elseif strcmp('turb',type)==1
        for k=1:length(t_lin)
            u_lin(k) = amplitude*rand(1);
            GenSpeed_reference(k) = SteadyOutput(2,posV_init);
        end
    elseif strcmp('steps',type)==1
        for k=1:length(t_lin)
            u_lin(k) = amplitude*floor(k*t_p/(t_end/num_steps))/(num_steps-1);
            GenSpeed_reference(k) =
SteadyOutput(2,posV_init+floor(amplitude*floor(k*t_p/(t_end/num_steps))/(num_s
teps-1)));
        end
    end

    if strcmp('wind',dist)==1
        wind_det(t_end, t_step, t_p, V_init, V_init+amplitude, num_steps,
freq, missalignment, type, DETfile);
        y_lin = lsim(sys_FAST(:,'horizontal hub-height wind
speed',posV_init),u_lin,t_lin);
        Tg_pert(1:t_end/t_p+1) = 0;
        Pitch_pert(1:t_end/t_p+1) = 0;
    elseif strcmp('pitch',dist)==1
```

```
        wind_det(t_end, 0, t_p, V_init, V_init, num_steps, 0, missalignment,
'lin', DETfile);
        y_lin = lsim(sys_FAST(:,'dPitch',posV_init),u_lin,t_lin);
        Tg_pert(1:t_end/t_p+1) = 0;
        Pitch_pert(1:t_end/t_p+1) = u_lin(:); %+ PitchAngles(posV_init);
    elseif strcmp('torque',dist)==1
        wind_det(t_end, 0, t_p, V_init, V_init, num_steps, 0, missalignment,
'lin', DETfile);
        y_lin = lsim(sys_FAST(:,'dTg',posV_init),u_lin,t_lin);
        Tg_pert(1:t_end/t_p+1) = u_lin(:); % + NomTorqueArray(posV_init);
        Pitch_pert(1:t_end/t_p+1) = 0;
    end

    lin_fast = zeros(size(y_lin,1),size(SYSTURB.outputname,1));
    for k=1:size(SYSTURB.outputname,1)
        lin_fast(:,k) = y_lin(:,k) + SteadyOutput(k,posV_init);
    end

    % SIM_FAST (SIMULINK)
    cd (FOLDER)
    input_fast = FSTfile; % archivo a ejecutar
    Read_FAST_Input
    sim('CL_simufast')
    cd ..\..\..
    V_sim_ramp = u_lin+V_init;

    % Figuras memoria
    numfig = numfig_init;

    figure(numfig);
    h(1)=subplot(3,1,1); hold all;
    if strcmp('wind',dist)==1
        plot(t_lin,u_lin+Windspeeds(posV_init),'color',col_int1); hold on;
        legend(legendTextDist);
        ylabel('V (m/s)');
        title('Wind Speed (m/s)');
    elseif strcmp('pitch',dist)==1
        plot(t_lin,u_lin,'color',col_int1); hold on;
        legend(legendTextDist);
        ylabel('\beta (rad)');
        title('Pitch Angle (rad)');
    elseif strcmp('torque',dist)==1
        plot(t_lin,u_lin,'color',col_int1); hold on;
        legend(legendTextDist);
        ylabel('T_g (N·m)');
        title('Torque (N·m)');
    end
    h(2)=subplot(3,1,[2,3]); hold all;

plot(t_sim_ramp,OutData(:,strncmp('GenSpeed',cellstr(SYSTURB.outputname),8)),'
color',col_int1); hold on;

plot(t_lin,lin_fast(:,strncmp('GenSpeed',cellstr(SYSTURB.outputname),8)),':','
color',col_int2); hold on;
    legend(legendText);
    ylabel('\Omega_g (rpm)');
    linkaxes(h,'x');
    xlabel('Time (s)');
    numfig=numfig+1;
```

```
    figure(numfig);
    h(1)=subplot(2,1,1); hold all;

plot(t_sim_ramp,OutData(:,strncmp('GenTq',cellstr(SYSTURB.outputname),5)),'col
or',col_int1); hold on;

plot(t_lin,lin_fast(:,strncmp('GenTq',cellstr(SYSTURB.outputname),5)),':','col
or',col_int2); hold on;
    legend(legendText);
    ylabel('T_g (kN·m)');
    xlabel('Time (s)');
    h(2)=subplot(2,1,2); hold all;

plot(t_sim_ramp,OutData(:,strncmp('BlPitch1',cellstr(SYSTURB.outputname),8)),'
color',col_int1); hold on;

plot(t_lin,lin_fast(:,strncmp('BlPitch1',cellstr(SYSTURB.outputname),8)),':','
color',col_int2); hold on;
    legend(legendText);
    ylabel('\beta (deg)');
    linkaxes(h,'x');
    xlabel('Time (s)');
    numfig=numfig+1;

    figure(numfig);
    h(1)=subplot(3,1,1); hold all;
    if strcmp('wind',dist)==1
        plot(t_lin,u_lin,'color',col_int1); hold on;
        legend(legendTextDist);
        ylabel('V (m/s)');
        title('Wind Speed (m/s)');
    elseif strcmp('pitch',dist)==1
        plot(t_lin,u_lin,'color',col_int1); hold on;
        legend(legendTextDist);
        ylabel('\beta (rad)');
        title('Pitch Angle (rad)');
    elseif strcmp('torque',dist)==1
        plot(t_lin,u_lin,'color',col_int1); hold on;
        legend(legendTextDist);
        ylabel('T_g (N·m)');
        title('Torque (N·m)');
    end
    h(2)=subplot(3,1,[2,3]); hold all;

plot(t_sim_ramp,OutData(:,strncmp('GenPwr',cellstr(SYSTURB.outputname),6)),'co
lor',col_int1); hold on;

plot(t_lin,lin_fast(:,strncmp('GenPwr',cellstr(SYSTURB.outputname),6)),':','co
lor',col_int2); hold on;
    legend(legendText);
    ylabel('Power (kW)');
    linkaxes(h,'x');
    xlabel('Time (s)');

end
```